Self configuring Bluetooth Mesh network algorithm

BSc Project
Group CS6-652
Communication Systems
Aalborg University
May 29, 2017

AALBORG UNIVERSITY
STUDENT REPORT

BSc Communcation Systems
Fredrik Bajers Vej 7
9220 Aalborg Ø

http://www.es.aau.dk

Title:
Self configuring Bluetooth Mesh network algorithm

BSc project:
Communication Systems

Project period:
Spring semester 2017
1. February 2017 - 30. May 2017

Project group:
CS6-652

Participants:
Daniel Edelgaard Serup
Emil Alexander Juul Hansen
Robin Jess Williams
Martin Hedegaard Nielsen

Supervisors:
Tatiana Kozlova Madsen
Rasmus Abildgren

Page number: 165
Date of completion: 30-05-2017

Abstract:

The purpose of this project is to design and implement a distributed algorithm which assigns relay functionality to a set of nodes in order to configure a mesh network of Bluetooth devices.
The configuration of such a network is an interesting topic for the upcoming Internet of Things.
The project starts by analysing the lower layers of the Bluetooth mesh protocol stack. Afterwards, different algorithms are analysed. For testing the different algorithms a Bluetooth mesh network simulator is designed from scratch.
In the simulator the performance of different relay distribution algorithms are compared to each other. A modified greedy algorithm favouring high connectivity is found to show reasonable results. It is therefore chosen to implement the algorithm on nodes in the network simulator.
The proposed algorithm showed that a Bluetooth mesh network with 200 devices is configured in 330.3 s on average. The found set of relay nodes resulted in a packet delivery ratio of over 90 % when network wide packages intensity are under 50 packet per second.

Resume

Dette projekt har til formål at se på hvordan konfigureringen af et Bluetooth Mesh netværk vil kunne automatiseres. Projektets emne er foreslået af, og lavet i samarbejde med, Samsung Denmark Research Center. En tidlig udgave af Bluetooth mesh standarden er stillet til rådighed af Samsung.

Projektet starter med en analyse af Bluetooth mesh standarden. Det er fundet at Bluetooth Mesh Standarden er lavet til billige lav-energi enheder som favoriserer et simpelt design over ydeevne. Visionen bag mesh netværket af Bluetooth enheder er at, det ville kunne lade sig gøre at forbinde en større mængde af enheder til internettet ved hjælp af Bluetooth.

For at alle enhederne i netværket kan kommunikere med alle andre enheder i netværket, er det nødvendigt at have nogle enheder som videresender beskeder, disse enheder kaldes relays.

Mængden af relays viser sig i projektet at være en betydelig faktor for netværkets ydeevne. Mængden af relays er derfor vigtigt at overveje når et sådanne netværk skal konfigureres. På baggrund af dette blev følgende problemformulering opstillet:

Hvordan kan relayfunktionalitet i et Bluetooth-netværk tildeles med henblik på ydeevne, og hvordan kan denne opgave ske automatisk.

Med denne problemstilling er det således ønsket at automatisere fordelingen af relayfunktionalitet således at ydeevne er favoriseret.

For at løse denne problemformulering er en netværkssimulator udviklet. Simulatoren er lavet i MATLAB af projektgruppen selv. Formålet med udviklingen at denne simulator er at kunne vægte fordele og ulemper for forskellige konfigureringsalgoritmer op mod hinanden.

Forskellige relay fordelingsalgoritmer er vurderet i netværkssimulatoren. En algoritme som 'grådigt' fordeler relay funktionalitet til enheder med mange forbindelser, er vurderet som den bedste algoritme til projektets formål.

Projektet giver derefter et forslag til hvordan den valgte algoritme kan implementeres på enheder i et netværk. Det foreslås at enheder i netværket skal starte med at afgøre hvilke, og hvor mange naboerenheder de har. Dette gøres over en tidsperiode hvor hver enhed sender konfigureringspakker med den fornødne information. Når enhederene i netværket har afgjort deres naboforhold, vil netværkets enheder hver for sig beslutte sig for hvilke enheder der skal være relayenheder.

Den implementerede algoritme er afslutningsvist simuleret. Her viser det sig at et netværk med 200 enheder kan opnå en pakke leverings rate på 90% efter i gennemsnit 330 sekunders konfigureringsprocess. Dette er ved en netværkstraffikintensitet på mindre end 50 pakker per sekund.

Preface

This report is composed by group CS6-652 as part of a bachelor project in wireless communication at Aalborg university.

Citations use IEEE style referencing. For this report two Bluetooth specifications are used. If nothing else is indicated, information about Bluetooth Low-Energy (BLE) and Bluetooth Mesh (BM) will always be from the Bluetooth Core v 5.0 Specification or the Bluetooth Core Mesh Specification [1][2].

Throughout the report, the name ideal and theoretical are used interchangeably when either node sets or algorithms are mentioned. This report denotes Byte as B and bit as b. Variable names for code are written in `typewriter font`. If a variable does not have a unit it is denoted with 1.

In the report, message and packet are used interchangeably to denote a physical transmission.

Aalborg University, May 29, 2017

Daniel Edelgaard Serup
<dserup14@student.aau.dk>

Emil Alexander Juul Hansen
<eajh14@student.aau.dk>

Robin Jess Williams
<rwilli14@student.aau.dk>

Martin Hedegaard Nielsen
<mhni14@student.aau.dk>

Glossary

ADV Advertisement. 5, 6, 26
AWGN Additive White Gaussian Noise. 9, 34, 38, 40, 48, 74, 96

BER Bit Error Rate. 9, 10, 38, 39, 40, 48
BL Bearer Layer. 14
BLE Bluetooth Low-Energy. v, 2, 3, 5, 7, 8
BLEP Block Error Probability. 39, 45, 46, 74, 76, 79, 80, 81, 82, 111
BM Bluetooth Mesh. v, 2, 3, 4, 5, 7, 10, 11, 12, 13, 14, 17, 18, 19, 20, 21, 22, 23, 25, 26, 28, 29, 33, 34, 35, 40, 43, 50, 52, 56, 57, 62, 98
BSIG Bluetooth Special Interest Group. 2, 98

CDF Cumulative distribution function. 85
CDS Connected Dominating Set. 50, 51, 52, 53, 55, 60, 68, 98
CPFSK Continous-Phase Frequency Shift Keying. 8

GATT Generic Attributes. 5, 6
GFSK Gaussian Frequency Shift Keying. 7, 8, 26, 28
GMSK Gaussian filtered Minimum Shift Keying. 8, 40

IoT Internet of Things. 1, 2, 3, 25, 98

LL Link Layer. 10, 11, 14, 35, 40, 41, 98
LOS Line Of Sight. 36, 37, 48
LTL Lower Transport Layer. 14

MAC Medium Access Control. 11, 12, 18, 20, 26, 28, 40, 43
MCDS Minimised Connected Dominating Set. 52, 55
MSK Minimum Shift Keying. 8

NL Network Layer. 4, 14, 15, 28, 34, 35, 44, 98
NLOS Non Line Of Sight. 36
NSP Network Setup Packet. 71, 72, 73

PBD Packets Before Decision. 84
PDR Packet Delivery Ratio. 3, 18, 19, 28, 31, 33, 34, 45, 60, 63, 65, 66, 68, 73, 74, 75, 76, 78, 79, 81, 82, 83, 84, 85, 86, 87, 90, 91, 93, 95, 96, 98, 99, 119, 142
PDU Protocol Data Unit. 14
PL Physical Layer. 6, 28, 35, 40, 98
PMF Probability Mass Function. 74
PPS Packets Per Second. 83, 84, 133, 135, 136

RSSI Received Signal Strength Indicator. 73, 75, 99

SINR Signal to Interference plus Noise Ratio. 35, 37, 38, 48
SNR Signal to Noise Ratio. 9, 37

TTL Time To Live. 14, 15, 16, 17, 26, 72

UTL Upper Transport Layer. 4

Contents

1 Introduction

In recent years the electronic industry has worked towards the Internet of Things (IoT). IoT means that every possible electrical device or appliance should be connected to each other and to the internet. Samsung Denmark Research Center has promised that all their products released after the year 2020, will be able to connect to the internet [3].

IoT introduces a lot of new devices with communication capabilities. All these devices have to be connected to a network while maintaining sufficient reliability and speed of the network. For mobile and battery powered devices, the communication system has to be power-efficient and flexible.

There are different approaches for creating such a network, using either dedicated wires or a wireless connection. A wired solution is a more expensive solution as the number of devices increase. For mobile and smaller cost efficient devices, a wireless system is preferred. Only wireless solution are considered in this project.

Wireless communication technologies can be split into short range and long range. Short range include technologies such as Wi-Fi, Bluetooth and ZigBee, all operating in the unlicensed 2.4 GHz band [4]. Long range include the licensed Global System for Mobile Communications (GSM) and NarrowBand Internet of Things (NB-IoT) and the non-licensed LoRa and SigFox. An overview of the mentioned communication technologies are illustrated on 1.1.

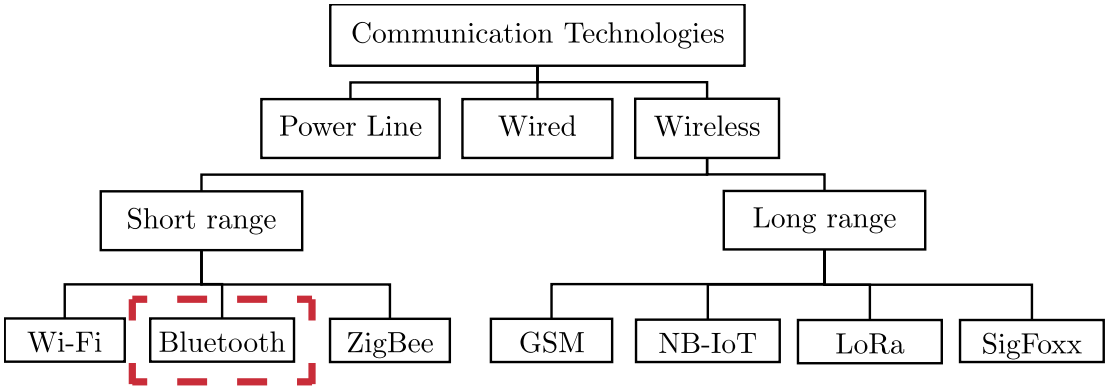


Figure 1.1: An overview of some of the communication technologies that can be used for IoT devices. This project has chosen to work with Bluetooth, marked by the red box.

One of the issues with IoT is that a lot of devices need to connect to a network. Therefore a dedicated network for handling traffic between IoT devices could be beneficial.

Depending on the use case either short range or long range can be used. In this project the use cases are "smart homes" and office environments. Therefore a short range communication standard is assumed.

The Bluetooth Special Interest Group (BSIG) is working on a Bluetooth Mesh (BM) network for short range low-power energy transmission, useful for battery powered IoT devices. BM builds on the Bluetooth Low-Energy (BLE) specification which is designed for cheap, simple, low power, and low data-rate devices. An early build of the BM specification is made available by Samsung Denmark Research Center for the project. BM is therefore the focus of this project.

This project begins by analysing the layers of the BM stack and how it performs in a generic smart home and office environment. The project continues by improving the performance of the network.

Analysis of Bluetooth mesh 2

As stated in Chapter 1 Bluetooth Mesh (BM) is to be used for the upcoming Internet of Things (IoT). The purpose of this chapter is to obtain technical knowledge required in order to understand the problems connected to the implementation of a BM network for the future IoT.

Firstly the Bluetooth Low-Energy (BLE) and BM technology are analysed. Secondly, an IoT use case is proposed and analysed.

The Bluetooth specifications is something that have been perfected over many years and is therefore very extensive. Since this is a student project with limited time and resources some delimitations have to be made before the technical analysis of the BLE and BM technology starts.

Only the parts of the BLE specification relevant for the understanding of BM is investigated.

The technical analysis aims to describe and analyse the layers of the BM protocol stack that influences the transmission reliability and Packet Delivery Ratio (PDR) of the network. Because of this security and encryption of the network is not described.

The analysis provides the foundation to make a problem statement which the rest of the project is focused around. Before the different layers of the BM architecture are analysed, some overall functionalities of a BM network is described in section 2.1.

2.1 Bluetooth Mesh network overview

Bluetooth Mesh (BM) is a network that is based on the Bluetooth Low-Energy (BLE) core specification and is designed for cheap, low-power, and low data rate devices. BM is meant for short range communication allowing devices to communicate to each other without the need for high power transmissions or complex circuitry. These criteria makes BM suited for the upcoming Internet of Things (IoT).

BM is suited for low data rate applications such as:

- An alarm clock signalling a coffee maker to start brewing, this way coffee is ready when one wakes up.
- A refrigerator transmitting its contents to a main hub. Which then orders the missing content.

- Door, window, and carport controllers for opening/closing, or locking/unlocking house entrances.
- Sensors placed around the house to monitor humidity, temperature, light, and movement for control of light bulbs and air conditioning system.
- Washing machine, dryer, dishwasher, and oven signalling tasks by sending a message to a main hub or a smartphone.

BM is however not suffice when high data throughput is necessary, for example in music or video applications such as wireless loud speakers and video streaming devices. This is because the BM only has a capacity of $1\,\mathrm{Mb\,s^{-1}}$(see Section 2.3.1).

In BM, a device which is connected to the mesh network is called a node. This naming is used throughout the project.

The overall structure of a BM network follows the general mesh network topology, with a few differences. In a general mesh network, all nodes in the network participate in the distribution of data within the network. This can either be done by flooding or other routing techniques. In BM flooding is used where not all nodes will necessarily relay packets.

BM introduces different roles to the mesh network, and the responsibility of data distribution is left to the nodes assigned the role of relay. An example of a BM network with the different roles is shown on Figure 2.1.

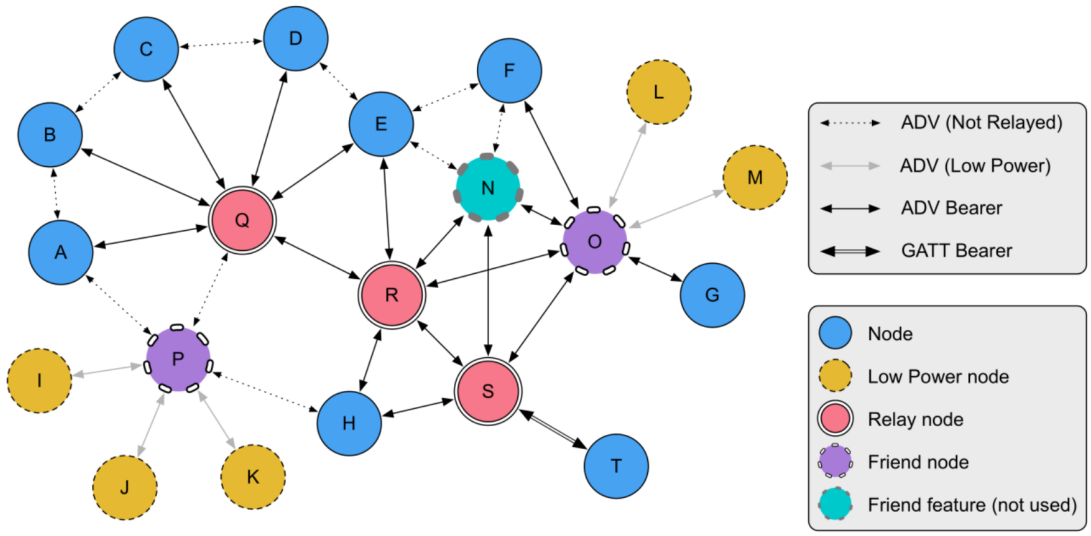


Figure 2.1: Example of a BM network giving an overview of the different roles possible in BM [2, page 43].

In BM four roles exists: generic nodes, low power nodes, relay nodes, and friend nodes. Nodes, relay nodes, and friend nodes are usually connected to a power line and are always listening for incoming transmissions. The low power nodes only occasionally talk with their friend node to conserve energy.

A node can have support for more than one role. Roles are assigned by the higher layers of the BM stack. The Relay feature is a Network Layer (NL) behaviour, where as friend

and low power features are Upper Transport Layer (UTL) behaviours.

As mentioned before, the responsibility of data distribution on the network is left to the nodes assigned the role of relay. A more detailed description of the relay node and its behaviour is given in Section 2.5.1 and Section 2.5.2.

Low power nodes are usually battery powered devices, and paired with a nearby friend node. With this pairing, the low power nodes is still able to communicate on the network without constantly listening for incoming messages. The friend node caches the messages sent to the low power nodes, and forwards the messages when it knows that the low power node is listening. Due to time constraints of this project, all nodes will be considered as regular nodes with relay functionality. Thus low power node and the friend feature will not be investigated any further.

Communication can happen using different bearers, the Generic Attributes (GATT) or the Advertisement (ADV) bearer. Nodes that are only able to communicate with the GATT bearer must be connected to another node supporting both bearers. The other node will then work as a proxy for the "GATT node", and the rest of the network will simply see the "GATT node" as part of the "proxy node". For this reason, all devices are assumed to use the ADV bearer.

2.2 Bluetooth Mesh stack overview

Bluetooth Mesh (BM) is a network that is based on the Bluetooth Low-Energy (BLE) core specification. This means that the Physical Layer and Link Layer of the BLE protocol stack is reused for BM.

The BM protocol stack has a total of nine layers. On Figure 2.2 the layers of the BM protocol stack is seen [2, page 26].

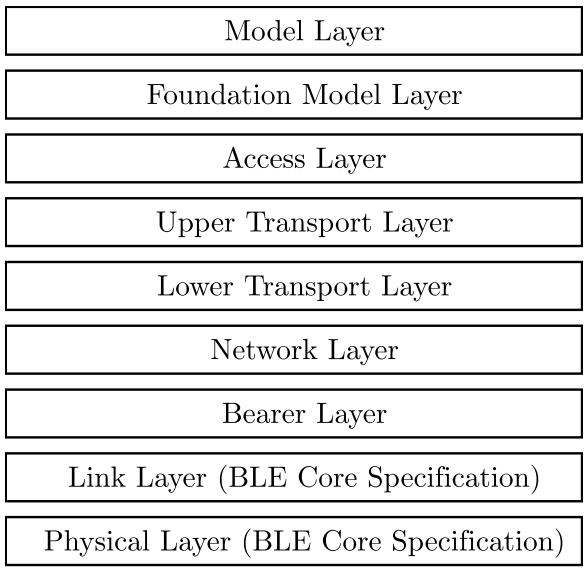


Figure 2.2: The layers of the BM protocol stack [2, page 26].

The overall functionality of the BM protocol stack layers are as follows [1, part A and B] [2, page 26-27]:

- The **Physical Layer** defines the modulation scheme, antenna operation, and provides services to the Link Layer.
- The **Link Layer** defines how the Physical Layer accesses the shared medium by handling which state the Physical Layer is in.
- The **Bearer Layer** defines which of the two bearers, Generic Attributes (GATT) or Advertisement (ADV) should be used. Since all nodes are assumed to use the ADV bearer. As described in 2.1, this layer is not described any further.
- The **Network Layer** defines how to encrypt packets, handles source/destination addresses, and implements the flooding routing scheme by handling the relay feature, TTL and caching of relayed messages.
- The **Lower Transport Layer** has the purpose of defining how to segment Upper Transport Layer packets in order to disassemble and reassemble Upper Transport Layer Protocol Data Unit's.
- The **Upper Transport Layer** main purpose is to authenticate, encrypt, and decrypt application data from the Access Layer
- The **Access Layer** defines and controls both the format of the application data and the encryption/decryption used in the Upper Transport Layer. Furthermore it validates the network and application keys of incoming packets.
- The **Foundation Model Layer** manages the mesh network by defining the Access Layer states and messages.
- The **Model Layer** defines models of typical user applications such as temperature and lighting sensors and controllers.

For the purpose of this project, only the Physical, Link, and Network Layer are deemed relevant. A more in depth description of these layers are given in the following sections.

2.3 Physical Layer

The Physical Layer (PL) has different behaviours depending on which bearer is used. Since it is assumed in 2.1 that all nodes use the Advertisement (ADV) bearer, only the ADV behaviour of the PL is analysed.

The PL is specified to be a multi channel broadcast medium on a radio frequency of 2.402-2.4835 GHz. There are 40 channels in total, each operating in their own 2 MHz band. The center frequencies of the bands is given by Equation (2.1) [1, page 2535].

$$f_c(k) = 2402\,\text{MHz} + k \cdot 2\,\text{MHz} \quad [\text{MHz}] \quad (2.1)$$

Where:

$f_c(k)$	is the k'th center frequency	[MHz]
k	is the band number $k \in \mathbb{N} \vert 0 \geq k \geq 39$	[1]

The channels are numbered from 0 to 39 where channel 37, 38, and 39 are called ADV channels. The ADV channels 37, 38, and 39 uses respectively band 0, 12, and 39. The ADV channels are used to communicate without having to form a standard Bluetooth master slave link. The ADV channels are located outside the most commonly used Wi-Fi channels (1, 6, and 11), and should therefore experience less interference [5].

Bluetooth devices are classified into three different power classes. Power class 1 is the class which allows for the maximum transmission output power of 100 mW = 20 dBm [1, page 326].

On Figure 2.3 the 2.4 GHz frequency spectrum is illustrated with the Bluetooth advertisement channels and the three commonly used Wi-Fi channels marked.

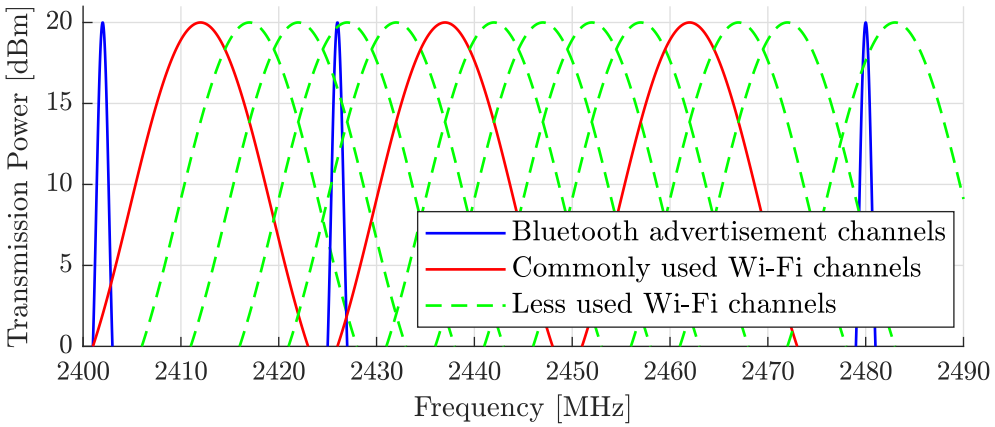


Figure 2.3: Illustration of the 2.4 GHz frequency spectrum with Bluetooth advertisement and Wi-Fi channels marked [6, section 4.3.1.2][5].

The rest of the channels from 0 to 36 are used for Bluetooth Low-Energy (BLE) frequency hopping and normal connection links. Any device can use the advertisement channels but may only send broadcast and not form pairwise connections on them [1, page 170].

A Packet send on the advertisement channel supports a payload of between 6 to 37 octets [1, page 209]. Furthermore a header of 10 B is added to the transmission [1, pages 2562 and 2567]. This gives a total length of 47 B.

2.3.1 Modulation characteristics

All communication in both BLE and Bluetooth Mesh (BM) is modulated with Gaussian Frequency Shift Keying (GFSK). GFSK is a modulation scheme where the bit stream is modulated into the frequency of the transmitted signal, and is therefore resistant to changes in signal power.

Every GFSK symbol contains a single bit of information, the symbol rate is therefore equal the bit rate. Thus $1\,\mathrm{sym\,s^{-1}} \implies 1\,\mathrm{b\,s^{-1}}$. This results in a relatively slower communication speed compared to other methods where symbols contains multiple bits. This modulation scheme however allows for simpler and cheaper transmitter and receiver designs, and a

sufficient spectral compactness.

All BLE devices must support a communication speed of $1\,\mathrm{Msym\,s^{-1}}$. Devices can however also have support for other speeds as well [1, page 2533]. However when an advertisement channel is used, the communication speed has to be $1\,\mathrm{Msym\,s^{-1}}$ [1, page 2567].

For the purpose of this project, it is assumed that all communication happens at a speed of $1\,\mathrm{Mb\,s^{-1}}$, and requirements only applicable for other communication speeds are therefore ignored.

GFSK is a type of Continous-Phase Frequency Shift Keying (CPFSK) where the bit-stream is filtered with a Gaussian filter, and the modulation index is given by Equation (2.2).

$$h = T_b \cdot (f_1 - f_2) = 0.5 \quad [1] \quad (2.2)$$

Where:

h	is the modulation index	[1]
T_b	is the duration of a bit	[s]
f_1	is the frequency representing a binary `1`	[Hz]
f_2	is the frequency representing a binary `0`	[Hz]

Specifically for Bluetooth the modulation index must be in the interval $h \in [0.45, 0.55]$, the bandwidth time product of the Gaussian filter must be $BT = 0.5$, and requirements for the minimum frequency deviation is also defined. To comply with the standard, devices must meet that $|f_c - f_1| \wedge |f_c - f_2| \geq 185\,\mathrm{kHz}$ when transferring with $1\,\mathrm{Mb\,s^{-1}}$ [1, page 2537].

In this project the modulation index for CPFSK is chosen to be $h = 0.5$ since it is the lowest modulation index allowing the CPFSK signals, representing binary `1` and `0`, to be coherently orthogonal [7, page 389].

CPFSK with $h = 0.5$ is also called Minimum Shift Keying (MSK). MSK basically works by changing the frequency with an interval of a bit duration T_b to indicate the value of the next bit in the sequence. The message signal for a symbol $m(t)$ can be described by Equation (2.3).

$$m(t) = \cos\left(2\pi \cdot \left[f_c + \frac{f_1 - f_2}{2} \cdot b\right] \cdot t + \theta(0)\right) \quad [\mathrm{MHz}] \quad (2.3)$$

Where:

$m(t)$	is the waveform for MSK symbol	[1]
f_c	is the carrier frequency	[MHz]
b	is 1 for binary `1` and -1 for binary `0`	[1]
$\theta(0)$	is phase of the signal at the start of the symbol	[rad]
t	is the time where $t \in [0; T_b]$	[s]

In the case of Gaussian filtered Minimum Shift Keying (GMSK), the sequence of values b is run through a Gaussian filter. Using a Gaussian filter with a bandwidth bit period product $BT = 0.5$, the phase difference between the carrier signal and $m(t)$, for a bit-sequence of [1 0 1 0 0 0 1 0 1 0 0 1 1 0 1 0], is illustrated on Figure 2.4. An example of a waveform for the first four bits is illustrated on Figure 2.5.

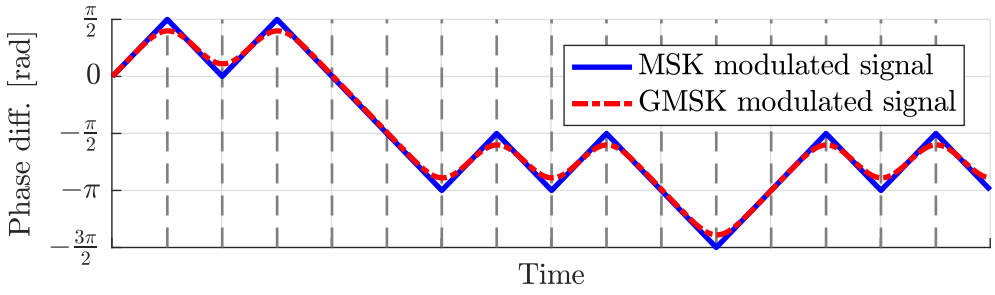


Figure 2.4: Graph of the phase difference between the carrier frequency and the message signal, for a bit sequence of [1 0 1 0 0 0 1 0 1 0 0 1 1 0 1 0], when modulating by GMSK or MSK. The vertical dashed lines indicates new bits. GMSK is done with a bandwidth time product $BT = 0.5$ and is shifted one symbol in time.

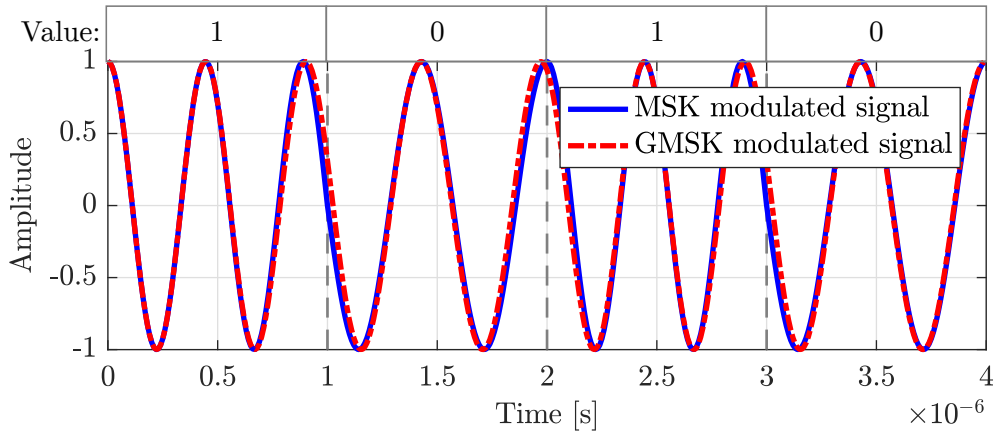


Figure 2.5: Graph of the waveform of GMSK and MSK modulated signals. A carrier frequency of $f_c = 2\,\text{MHz}$ is used to modulate the sequence [1 0 1 0] with a bit rate of $BR = 1\,\text{Mb}\,\text{s}^{-1}$. GMSK is done with a bandwidth time product $BT = 0.5$ and is shifted one symbol in time.

Bit Error Rate

When a signal travels through a medium it will experience different kinds of noise and interference which will effect the original signal.

In order to quantify how well a modulation scheme can carry data, Bit Error Rate (BER) is found as a function of Signal to Noise Ratio (SNR). The function is dependent on the channel used. For the purpose of this project it is assumed that an Additive White Gaussian Noise (AWGN) channel will be sufficient for calculating a BER for the modulation scheme.

AWGN is chosen because all transmissions from Bluetooth devices have been whitened [1, page 2601]. This means that transmissions from other Bluetooth devices in the same spectrum will simply be seen as AWGN to the receiver.

By assuming an AWGN channel, it is assumed that the noise in the spectrum are of equal power and with a normal distribution.

For an AWGN channel the BER can be found using the equation published by Murota and Hirade [8]. The equation is a theoretical model of how the BER should be calculated and is given by Equation (2.4).

$$P_e(\gamma) = \frac{1}{2}\text{erfc}(\sqrt{\alpha\gamma}) \quad [1] \quad (2.4)$$

Where:

P_e	is the probability of an error	[1]
γ	is the received signal energy to noise density ratio	[1]
erfc	is the complimentary error function	[1]
α	is a parameter constant	[1]

The expression erfc(x) is the complimentary error function given by Equation (5.7).

$$\text{erfc(x)} = 1 - \text{erf}(x) = \frac{2}{\sqrt{\pi}} \int_x^\infty \text{e}^{-t^2}\text{d}t \quad (2.5)$$

Combining Equation (2.4) and Equation (5.7) yields an expression linking the signal energy-to-noise density ratio to the BER is found as described in Equation (2.6).

$$P_e(\gamma) = \frac{1}{2} \cdot \frac{2}{\sqrt{\pi}} \int_{\sqrt{\alpha\gamma}}^\infty \text{e}^{-t^2}\text{d}t = \frac{1}{\sqrt{\pi}} \int_{\sqrt{\alpha\gamma}}^\infty \text{e}^{-t^2}\text{d}t \quad [1] \quad (2.6)$$

In Equation (2.6), the parameter constant, α is a is chosen to be 0.68 for GMSK [8].

2.3.2 Advertisement events

When transmitting on the physical channel a device initializes an advertisement event, which consists of three advertisements sent on the each of the advertisement channels. The device will first send the packet on channel 37, then on channel 38, and finally on channel 39 [1, page 2614]. A device may end its advertisement event at any time during the advertisement event. Any device can start its own advertisement event while another device is transmitting [1, page 2608].

An illustration of two advertisement events can be seen on Figure 2.6.

During an advertisement event, the time between the start of one advertisement to the start of the next advertisement shall be $\leq 10\,\text{ms}$ [1, page 2614].

Since any devices can start transmitting while another device is transmitting, collisions can occur as described in 2.4.2. The way channelisation and advertisement works makes it possible to have multiple devices transmit at the same time without having collisions. In the physical layer, each message is transmitted on all three advertisement channels. In the case where more devices are scanning on different channels during an advertising event, some nodes can still receive the packet even if a channel is occupied by noise. The advertisement scheme therefore has some redundancy, and a packet can still reach the final destination if there is noise on a channel.

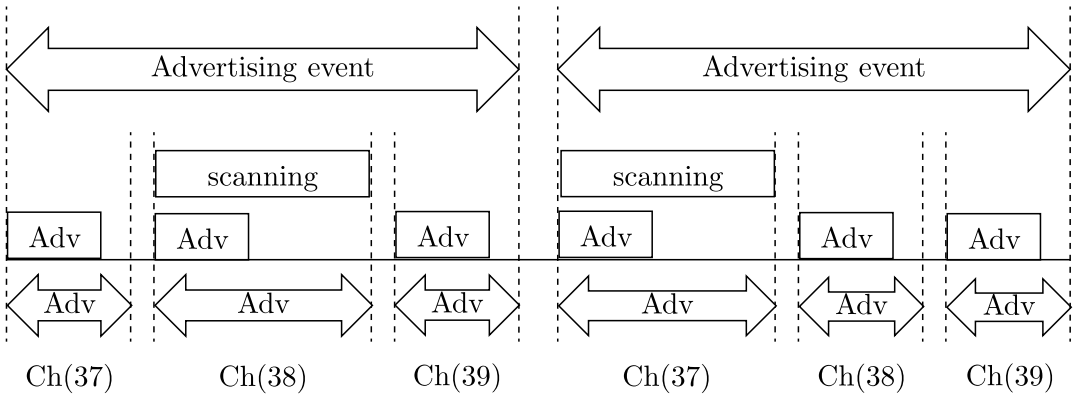


Figure 2.6: The advertisement sending scheme where the different channels are represented by Ch(37), Ch(38), and Ch(39). Inspired by [1, page 170 - Figure 1.3].

2.4 Link Layer

The second layer in the Bluetooth Mesh (BM) protocol stack is the Link Layer (LL) which handles data transmissions, establishes connections, and controls the state of the devices.

For establishing connections and for normal operation the link layer has five different states [1, page 2553]:

- Standby State
- Advertising State
- Scanning State
- Initiating State
- Connection State

Because of these five states, the LL can be understood as a state machine. The LL state machine is only able to be in one active state at the same time. However multiple LL state machines may exist in a node [1, page 2553].

For BM only the Standby State, Advertising State and Scanning State are used. Therefore only those three states will be further described.

Because of how BM works no Medium Access Control (MAC) regulations are present in the link layer, an analysis of the consequences this has on the network is made in Section 2.4.2.

Standby State

The Standby State can be entered from all the other states. In this state, no packets are sent or received [1, page 2553]. The Standby State is the default state of the LL [1, page 2608].

Advertising State

The Advertising State is used for quick packet exchange without acknowledgement, for the purpose of broadcasting data. The Advertising State sends data on all three advertising channels as described in Section 2.3.

The Advertising State can only be entered from the Standby State [1, page 2553].

A node can leave the Advertising State before the data has been broadcast on all three advertising channels as described in Section 2.3.2 and [1, page 2608].

Scanning State

In Scanning State, a node scans one of the three advertisement channels for advertisements from advertising nodes. This way a scanning nodes can receive data from an advertising node without establishing a connection first.

The scanning state can only be entered from the Standby State [1, page 2553].

2.4.1 Network setup

In a BM network, all modes in the network should be in the Scanning State as default. If one node wants to broadcasting a message it has to change its state to the Advertising State.

If a Relay node in the Scanning State receives a message which it needs to relay it has to change to the Advertising State.

When a node changes state to the Advertising State, it is able to broadcast a message to all nodes in the Scanning State who are within range.

The node only enters the Standby state permanently if it does not want to participate in the network.

These different shifts adds time delays to the system because it can not shift from advertising to scanning but has to go through the Standby State.

2.4.2 Medium Access Control

With BM no MAC is implemented. This means that nodes will not listen before broadcasting starts, the devices simply starts transmitting whenever nodes have something to transmit. BM is therefore uncoordinated. This having the consequence that if a packet loss occurs, the packet is not retransmitted unless some connection-oriented protocol is added at a higher layer.

Compared to other schemes where MAC regulations are in place, BM has a higher probability of collisions and overall lower reliability due to the lack of collision detection.

The purpose of BM is not to have the most reliable and collision free communication. However, to have a cheap and simple system which is sufficient for applications where high transmission speed is not necessary.

For determining the probability of collisions in the BM network some assumptions have to be made. It is assumed that only one channel is used to send a packet.

Only self interference is accounted for, this means that collisions happen in the network if two or more nodes within range of one another broadcast at the same time. An example of a collision is shown on Figure 2.7.

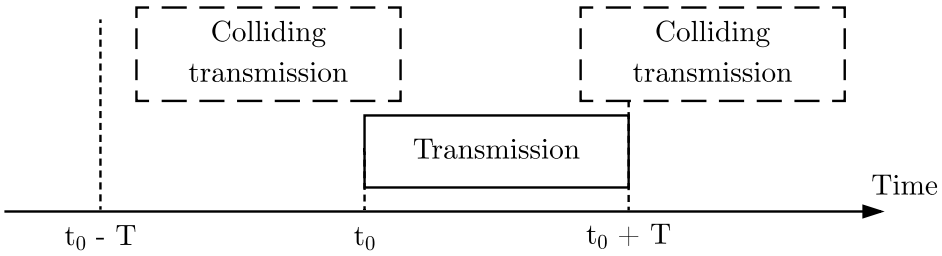


Figure 2.7: Diagram showing collisions that can occur with BM. If another transmission begins within the vulnerable period from $t = t_0 - T$ to $t = t_0 + T$, a collision will occur.

From Figure 2.7 it is seen that a collisions occur if another transmission is begun within the vulnerable period $\Delta t_v = (t_0 + T) - (t_0 - T) = 2T$, where T is the duration of one packet transmission. This statement assumes all transmissions have the same duration.

Knowing the minimum transmission speed of a Bluetooth node to be $v_t = 1\,\mathrm{Mb\,s^{-1}}$ [1, page. 2533] and the maximum packet size to be 47 B [1, pages 2562 and 2567]. The worst case vulnerable time frame is therefore given by Equation (2.7)

$$\Delta t_v = 2 \cdot T = 2 \cdot \frac{n_{bit}}{v_t} = 2 \cdot \frac{47\,\mathrm{B}}{1\,\mathrm{Mb\,s^{-1}}} = 0.752\,\mathrm{ms} \tag{2.7}$$

Where:

Δt_v	is the vulnerable period	[s]
n_{bit}	is the number of bits in a transmission	[b]
v_t	is the transmission speed	$[\mathrm{b\,s^{-1}}]$
T	is the transmission duration	[s]

Equation (2.7) yields that a Bluetooth node with the minimum required speed will have a vulnerable period of 0.752 ms when send a packet of maximum length.

To consider the probability of a collision free transmission, it is assumed that a transmission is occurring. If a transmission is occurring, a collision will happen if another device begins transmission within the vulnerable period Δt_v. The probability of a collision free transmission is therefore the probability of no other transmissions beginning within the vulnerable period, as written in Equation (2.8).

$$Pr(\text{no collision}) = Pr(\text{no other transmissions within } \Delta t_v) = q \quad [1] \quad (2.8)$$

Where:

q	is the probability of no packet collisions	[1]

The value of q can be determined experimentally, through simulations, or derived mathematically. The value of q is however based on the exact network specifications, and will therefore not be determined in this section. The value of q is approximated under a range of assumptions in Section 2.6.1.

2.5 Network Layer

The Network Layer (NL) is the third layer of the Bluetooth Mesh (BM) stack. The NL is responsible for:

- Serving as an interface between the Bearer Layer (BL) and the Lower Transport Layer (LTL).
- Implementing the flooding routing scheme by:
  - Administrating the relay feature
  - Network packet caching
  - Modifying the Time To Live (TTL) counter
  - Deciding whether a packet is relayed or not.
- Addressing of packet senders and receivers

The NL interfaces with the LTL and the Link Layer (LL) through the BL. Packets sent between the NL and LTL has a size of 1-16 B and contains some payload as well as overhead added by the above layers. Packets sent between the NL and the LL have a size of 14-33 B and contains the data from or to the LTL as well as addressing, TTL, and other control information. The length of the NetMIC is 64 b for control messages that are not authenticated at the upper transport layer. [2, page 121]

The contents of a NL Protocol Data Unit (PDU) as well as a short description is given in Table 2.1. [2, page 52]

Table 2.1: The BM network PDU format [2, page 52].

Field Name	**Bit size**	**Note**
IVI	1	Least significant bit of IV Index
NID	7	Value derived from the NetKey used to identify the Encryption Key and Privacy Key used to secure this PDU
CTL	1	Network Control
TTL	7	Time To Live
SEQ	24	Sequence Number
SRC	16	Source Address
DST	16	Destination Address
TransportPDU	8 to 128	Lower Transport Protocol Data Unit
NetMIC	32 or 64	Message Integrity Check for Network

BM uses three types of addresses: Unicast, Virtual, and Group Addresses [2, page 49-51]. A node containing multiple elements[1] can have multiple addresses. A unicast address is a unique address pointing at one specific element in the entire network. Group and virtual addresses can be used to address a packet to a group of elements in the network, e.g. all light bulbs in the network. BM allows for 32767 unicast, 16384 group, and 16384 virtual addresses [2, page 32].

The relay feature, network packet caching, TTL counter, and relay decision protocol is studied to give an understanding of how the NL works. The flooding scheme that the NL implements is evaluated afterwards.

2.5.1 Relay feature, TTL, and network packet caching

The Relay feature, TTL, and network packet caching will now be analysed.

Relay feature The relay feature of a node can be in three different states. The relay feature can either be supported but not enabled, supported and enabled, or not supported [2, page 149]. The support of the relay feature is decided by the manufacturer of the device, and there is no standard for how this is decided.

When the relay feature is enabled, the node continuously scans for incoming messages. If a message is received the node will decide if the packet should be relayed. The relay decision process will be described in Section 2.5.2.

Time To Live The TTL defines how many hops a message is allowed to take in a network. The value of TTL ranges from 127 to 1, with 0 being a special case.

If a packet has a TTL $\in [2, 127]$, the packet can be relayed and TTL is subtracted by one. If a packet has a TTL $\in \{1, 0\}$, the packet must not be relayed. In the special case where TTL $= 0$, the receiver knows that the packet is sent from a neighbour which can be used to gain knowledge regarding the topology of the network. [2, page 53-54]

Network packet caching Every node must have a packet cache containing either the full contents or parts necessary for tracking of at least two most recently received packets [2, page 58]. If a received packet is already contained in the network packet cache, it is discarded.

The implementation of the network packet cache together with the TTL ensures that no packet will exist in the network for ever, and that no node relays the same packet multiple times, thus lowering the overall traffic within the network.

[1] an example of a node having multiple elements could be a mobile phone running multiple applications all using bluetooth. [1, page 2553]

2.5.2 Relay decision protocol

Once the NL receives a packet, the layer will go through a decision protocol as illustrated in Figure 2.8.

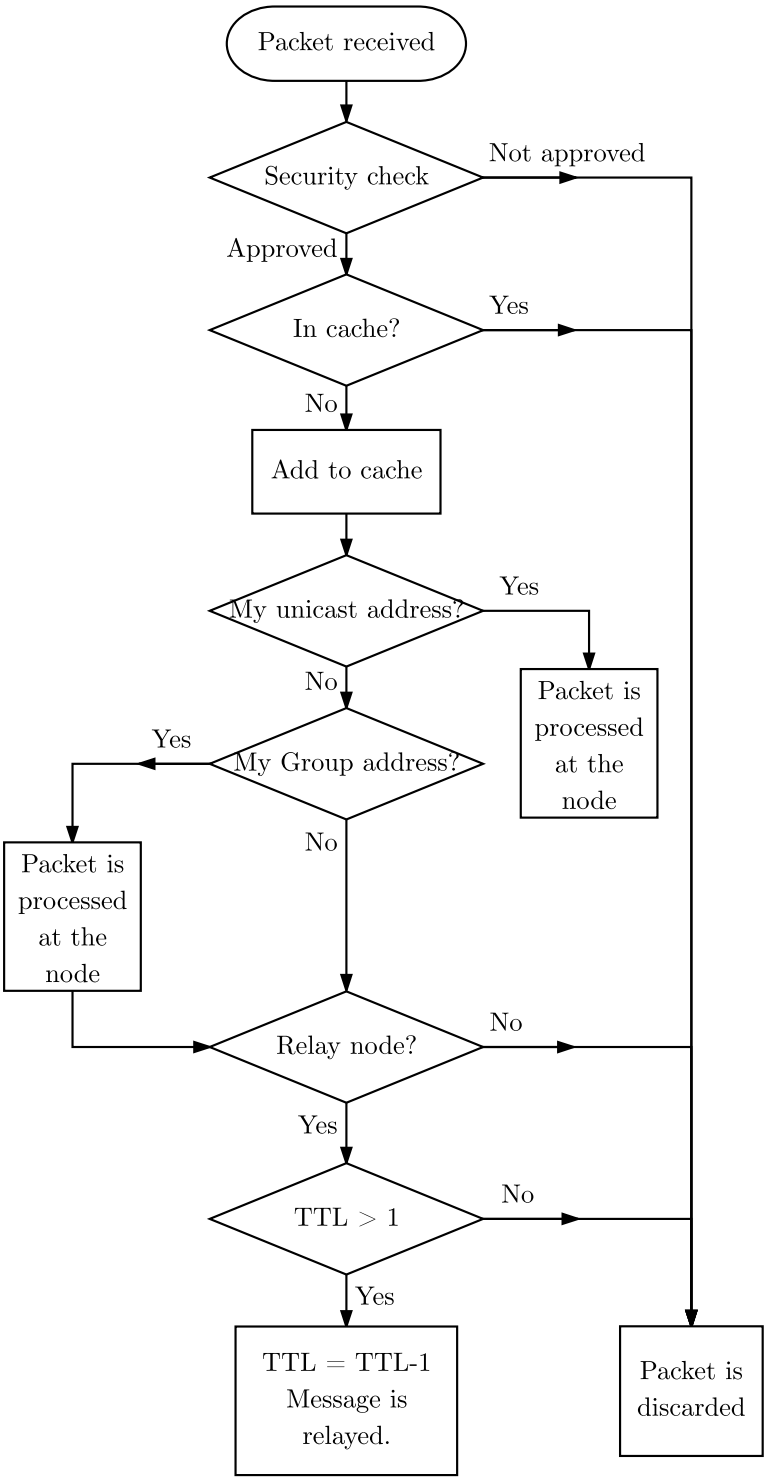


Figure 2.8: Flow diagram showing the Network layer protocol to receiving a network packet. The network layer also does encryption and decryption, this is however not shown on the figure [2, page. 57].

As shown on Figure 2.8 the node node first validates and authenticates the security key of the packet. Afterwards the node checks if the packet is in the message cache list. If the packet is addressed to the node itself or a group address the node is subscripted to. Lastly the node decides if the packet should be relayed. If the node is a relay node and the packet has a TTL of more than 1, the packet is relayed.

2.5.3 Flooding in a wireless network

The relay feature, TTL, network packet caching, and the associated relay decision protocol effectively implements flooding to transport packets throughout the network. For this reason flooding is studied in this section.

To investigate the advantages of flooding, as well as the problems that follows, an example of a transmission from a node A to a node B is considered. Node A transmits a packet with node B as the destination. The packet is then transported throughout the whole network by the relay nodes assuming that the packet is not lost under way.

Figure 2.9 illustrates how a packet send from A to B might travel through the network.

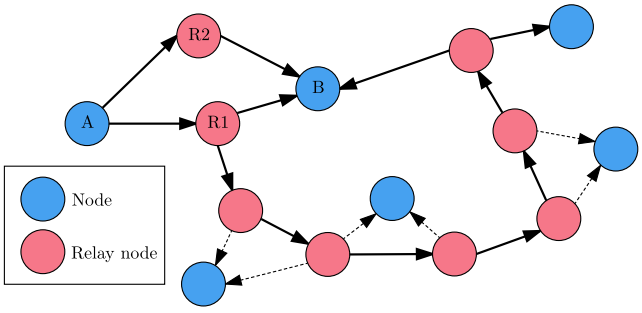


Figure 2.9: Example of flooding where a packet is transmitted from node A to node B. The packet is transmitted through the whole network even though the destination is two "jumps" away. The arrows denote the direction the packet travels in the network.

The packet travels all possible paths from node A to the destination node B, creating a redundant system. If a path fails, the packet might still reach its destination though an alternative paths. This redundancy is an advantage of the flooding scheme.

The multi-path routing created by the flooding scheme comes at the cost of increased traffic throughout the network. If no countermeasures are implemented the increased amount of traffic might result in a broadcast storm. A broadcast storm is when a communication system is overwhelmed by traffic, causing all communication to fail. In BM the previous mentioned TTL and network packet caching reduces the risk of a broadcast storm, by reducing the probability that packets are sent twice in the network.

When analysing the network setup on Figure 2.9 it can be seen that the time between packet receipt and relaying should not be the same for all relay nodes to avoid assured collision. If the time between packet receipt and relaying is the same for all relay nodes, relay nodes R1 and R2 would retransmit the packet at approximately the same time

resulting in a collision. Some countermeasures have to be implemented to ensure that R1 and R2 will not both relay the packet immediately.

For networks with short and few packets, flooding makes for a viable solution due to the absence of overhead. Alternatives such as routing schemes uses control packets to keep track of the network topology. The amount of control packets can get relatively big compared to the actual data, resulting in limited gain compared to the increased complexity of the devices.

2.6 Bluetooth Mesh use case analysis

As stated in Section 2.4.2 no Medium Access Control (MAC) regulations are implemented in Bluetooth Mesh (BM), therefore the probability of collisions might be high depending on the traffic in the network. In this section, two use cases of BM networks are analysed. Two generic use cases are arbitrarily chosen: a house of 130 m^2 and a office of 560 m^2. The two use cases are used in this section to determine whether a network built on BM could perform sufficiently for smart-home and office environments, and how relays should be distributed to maximise Packet Delivery Ratio (PDR).

Illustrations of the two use cases are seen on Figures 2.10 and 2.11. On the figure humidity, light, movement, and temperature sensors are placed in clusters around the buildings as low power nodes. Sensors for detecting whether the outer doors and windows are closed and locked are connected to the network. Kitchen appliances, PCs, and tablets are also represented as network nodes. A node for an air conditioning system, a BM to internet

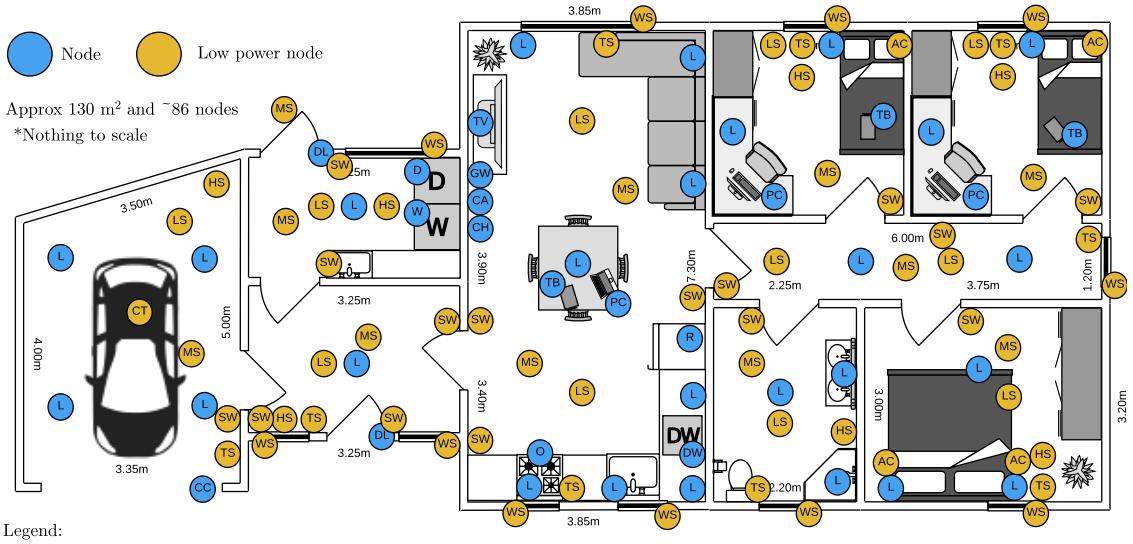


Figure 2.10: Floor plan of a small house with Bluetooth mesh nodes marked. The floor plan contains 101 nodes, where 58 are low power nodes and 43 are ordinary nodes.

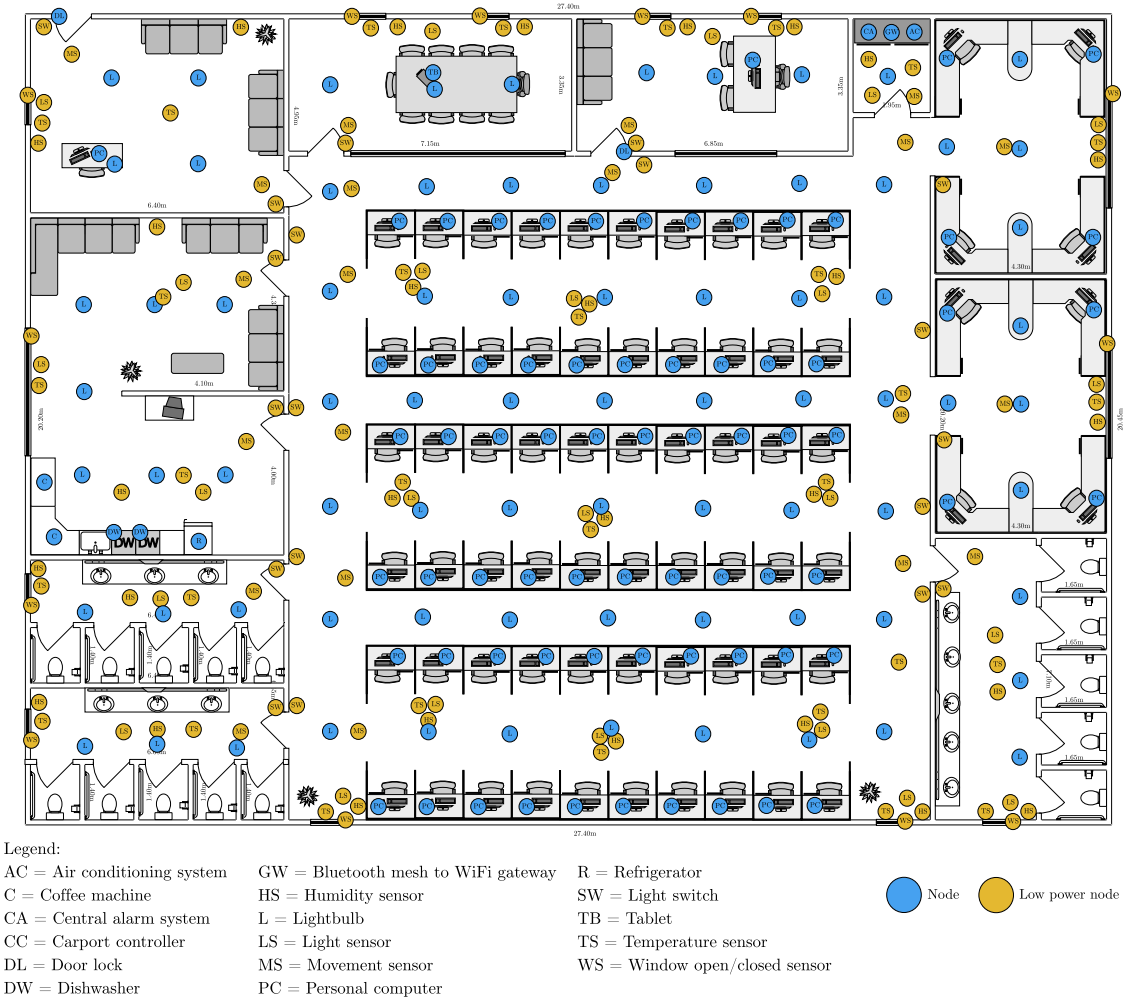


Figure 2.11: Floor plan of an office with Bluetooth mesh nodes marked. The floor plan contains 294 nodes, where 136 are low power nodes and 158 are ordinary nodes.

gateway, and a central alarm system is also connected. All light bulbs are given the status of a node as well.

From Figure 2.10 it is counted that a house of approximately 130 m^2 could easily be imagined to contain 101 Bluetooth devices which would benefit from being connected to a mesh network. In the single floor office plan as seen on Figure 2.11 a total of 294 nodes is counted. If the office was bigger and had multiple floors it would quickly become a 1000+ node network, dependent on the implementation multiple subnets[2] might also be the case.

2.6.1 Performance analysis

In a BM network the PDR is dependent on a lot of different parameters. It is among others dependent on the sensitivities, behaviour, and positions of the individual nodes. A colliding transmissions does not necessarily result in lost packets, and packet loss can still

[2]Different subnets can be created in a building, for example if some nodes have different security requirements than others.

happen during collision free transmissions.

A complete mathematical model of a real BM network is deemed out of the scope of this project. Therefore a delimited scenario of reality is analysed instead. The purpose of this analysis is to determine whether a BM network as described in Sections 2.4 and 2.5, could make for a sufficient communication system for the two proposed use cases. Is it feasible to communicate using a the BM standard, which has no MAC regulations and uses simple flooding?

In this analysis, transmissions are split into two categories; injection and relaying. An injection is a transmission of a new packet into the network and can be done by all nodes. A relaying is a retransmission of a packets which has already been injected into the network and can only be done by relay nodes.

For the first part of this analysis all nodes are assumed to be within reach of each other, meaning that there is no need for relay nodes. By having no relay nodes, the overall activity in the network is significantly reduced due to packets not being relayed. Since all nodes are within reach of each other, it is assumed that all collisions will result in lost packets. Furthermore, the frequency hopping nature of BM is also omitted for the sake of simplicity.

Due to these simplifications the first part of this analysis only considers injections[3]. The probability of successfully injecting a packet into the network $q_i(nn, r)$, is dependent on the number of neighbouring nodes[4], nn, and the number of repetitions, r. If injections are repeated, one or more transmissions (in one set of injection and repetitions) has to be collision free, for the injection to be successful. Equation (2.9) states this probability.

$$Pr(\text{successful injection}) = q_i(nn, r) = 1 - (1 - q_{cfi}(nn, r))^{r+1} \quad [1] \quad (2.9)$$

Where:

$q_i(nn, r)$	is Pr(successful injection)	[1]
$q_{cfi}(nn, r)$	is Pr(collision free injection)	[1]
nn	is the quantity of neighbouring nodes	[1]
r	is the number of repetitions	[1]

Equation (2.8) from Section 2.4.2 states that the probability of collision free transmission given a node is transmitting is equal the probability of no neighbouring nodes transmitting within a vulnerable period Δt_v. This probability $q_{cfi}(nn, r)$ is given in Equation (2.10). The repetitions are modelled as new independent injections. The maximum number of repetitions allowed is dependent on the nodes injection frequency, since the BM standard dictates a node should not send[5] more than 100 packets in a 10 s moving window [2, page 102].

[3] An injection is defined to be when a node sends at packet which is dependent on a relay to pick up.

[4] A neighbouring node is defined as a node within range of a given node

[5] Meaning that a node should not send more than 10 transmissions every second of packets with itself as the source. This requirement from the BM specification does not affect relaying packets from other nodes.

$$q_{cfi}(nn, r) = (1 - p)^{nn \cdot (1+r)} \quad [1] \quad (2.10)$$

Where:

p	is Pr(A node starting transmission within Δt_v)	[1]

The vulnerability period is given as $\Delta t_v = 0.752\,\text{ms}$, as calculated in Section 2.4.2 Equation (2.7). Assuming that all nodes injects one new packet every second, the probability p is given by Equation (2.11).

$$p = \frac{\Delta t_v}{1\,\text{s}} = \frac{0.752\,\text{ms}}{1\,\text{s}} = 0.752 \cdot 10^{-3} \quad [1] \quad (2.11)$$

Where:

Δt_v	is the vulnerable period	[s]
p	is Pr(A node starting transmission within Δt_v)	[1]

Hence $q_i(nn, r)$ is given as Equation (2.12).

$$q_i(nn, r) = 1 - \left(1 - \left(\left(1 - 0.752 \cdot 10^{-3}\right)^{nn \cdot (r+1)}\right)\right)^{r+1} \quad [1] \quad (2.12)$$

Since all nodes are within range of each other, the probability of successful injection is equal the packet arrival probability. Furthermore, the quantity of nodes in the network n is equal the amount of neighbouring nodes of any node nn plus one ($n = nn + 1$). By this, the packet arrival probability is given by Equation (2.13).

$$Pr(\text{packet arrival}) = q_a(n, r) = q_i(n - 1, r) \quad [1] \quad (2.13)$$

Where:

n	is the amount of nodes in the network	[1]

Figure 2.12 shows $q_a(n, r)$ in respect to the amount of nodes in the network n and the number of repetitions r. From Figure 2.12 it is seen that with the same amount of nodes as the house (101) and office (294) environments, the packet arrival probability is $q_a(101, 0) = 0.93$ and $q_a(294, 0) = 0.80$ respectively if injections are not repeated. In these two cases, the probability $q_a(n, r)$ can be increased by repeating the packet injection.

When injections are repeated once ($r = 1$), the packet arrival probability $q_a(n, r)$ is increased if the quantity of nodes $n < 600$. When $r = 1$, the probability $q_a(n, r)$ is increased by up to 0.0721. For $r = 2$, the probability $q_a(n, r)$ is increased by up to 0.0917, beating $r = 1$. Increasing r even further gives diminishing returns especially compared to how much worse $q_a(n, r)$ becomes as n increases. If the network is sparse, $r = 1 \vee 2$ is seen to result in a considerable increase in $q_a(n, r)$.

For high n it is seen that introducing repetitions only decreases $q_a(n, r)$. Repetitions should therefore be avoided in dense networks. If $r = 0$, the probability $q_a(1000, 0) = 0.47$ which could be sufficient for controlling an air conditioning system with an expected feedback every two seconds on average. For more complex data transmissions with cascading connected packets, this low a $q_a(n, r)$ might not be sufficient.

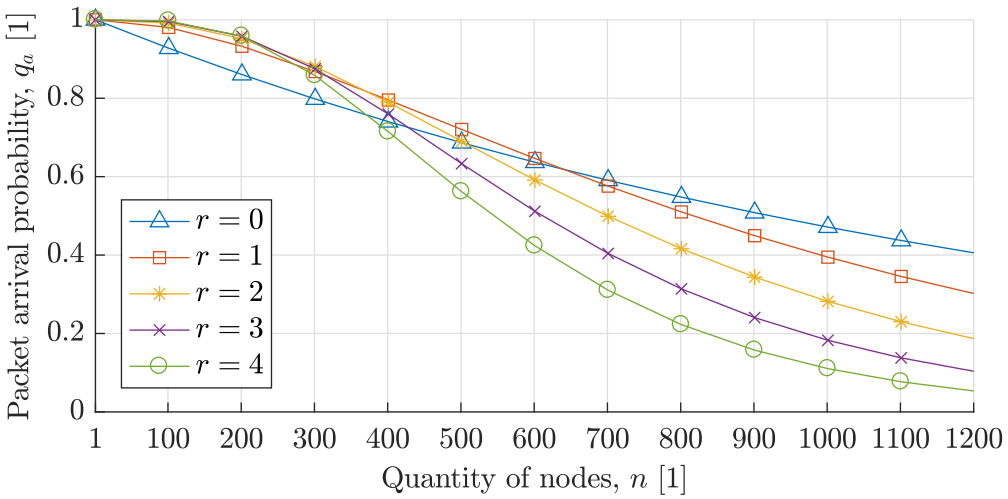


Figure 2.12: Graph showing the packet arrival probability q_a in respect to the quantity of nodes n and the number of repetitions r.

Relays are an essential part of a BM network, and the network is left unconnected without them. So before concluding this analysis, the effect of introducing relays into the network is considered.

2.6.2 Effect of varying relay amount

To analyse the effect of varying the quantity of relays in a BM network, a simplified network as illustrated in Figure 2.13 is considered.

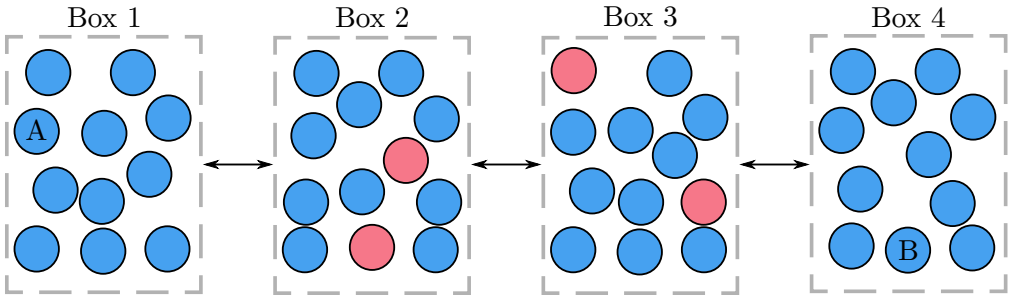


Figure 2.13: Diagram of a simplified BM network. Nodes in a dotted box can communicate with nodes in the same box, as well as nodes in the adjacent dotted boxes, as indicated by the arrows. Blue circles are ordinary nodes, red circles are relay nodes. The quantity of relay nodes in a box nr and injections per second from a box f_I is varied in the analysis. There are no relay nodes in the right- and leftmost boxes.

It is assumed that nodes within a dotted box is in range of all other nodes in the same box, as well as all nodes in the adjacent dotted boxes. There are a total of n nodes in every box, collectively injecting f_I packets per second. There can be one or more relays within a box, and the quantity of relay nodes in a box is denoted nr. There is the same amount of nodes in all boxes. A relay node does injections and relay transmissions while ordinary nodes only injects.

Node A in the leftmost dotted box transmits a packet destined for a node B in the rightmost dotted box. The packet must therefore be relayed through two dotted boxes to reach its destination.

The vulnerability period is set to be the maximum possible vulnerability period (see Section 2.4.2) of $\Delta t_v = 0.752\,\text{ms}$, and all collisions are assumed to cause packet loss.

To simplify this analysis, it is assumed that all relaying happens independently of injections. The frequency hopping nature of BM is still omitted in this analysis.

As mentioned in Section 2.6.1, this analysis categorises transmissions in two types: injection and relaying. These two types of transmissions have different probabilities of successfully reaching the adjacent boxes. This part of the analysis does not consider repetitions of injections. Due to the assumptions made, the probability of a packet being collision free, dependent on the traffic intensity ti in packet/s, is given by Equation (2.14).

$$q_{cft}(ti) = \left(1 - \frac{\Delta t_v}{1\,\text{s}}\right)^{ti \cdot 1\,\text{s}} = (1 - 0.752 \cdot 10^{-3})^{ti \cdot 1\,\text{s}} \quad [1] \quad (2.14)$$

Where:

$q_{cft}(ti)$	is Pr(a transmission being collision free)	[1]
Δt_v	is the vulnerable period	[s]
ti	is the traffic intensity	$[\text{s}^{-1}]$

Injection: With no repetitions, the probability of successfully injecting a packet is simply $q_{cft}(ti)$, as written in Equation (2.15).

$$q_i(ti) = q_{cft}(ti) \quad [1] \quad (2.15)$$

Where:

$q_i(ti)$	is Pr(successful injection)	[1]

Relaying: For relaying, the probability of successfully reaching the adjacent boxes is different. Since all nodes within a box are within range, all relays in the box will receive a packet if one of them does. This means that if a packet successfully reaches a box, all relays will try to relay to the adjacent boxes. The probability of reaching the adjacent boxes is therefore the probability of one or more of the individual transmissions being collision free. This probability is given by Equation (2.16).

$$q_r(ti, nr) = 1 - (1 - q_{cft}(ti))^{nr} \quad [1] \quad (2.16)$$

Where:

$q_r(ti, nr)$	is Pr (a packet begin relayed successfully)	[1]
nr	is the quantity of relays in a box	[1]

Traffic intensity: Since the probability of a successful transmission is dependent on the traffic intensity, it is estimated. If node A transmits a packet P to node B, the best case scenario would be if all other transmissions collide, leaving P as the only packet being relayed. The worst case scenario would be if all other packets are successfully injected and relayed by all relay nodes.

The number of transmissions made in the boxes, dependent on the injection frequency of a box f_I, in the best and worst case scenarios are given in Equations (2.17) to (2.20). Due to symmetry, $ti_1 = ti_4$ and $ti_2 = ti_3$.

$$ti_{1,\text{bestCase}} = f_I \quad [\text{s}^{-1}] \quad (2.17)$$

$$ti_{2,\text{bestCase}} = f_I \quad [\text{s}^{-1}] \quad (2.18)$$

$$ti_{1,\text{worstCase}} = f_I \quad [\text{s}^{-1}] \quad (2.19)$$

$$ti_{2,\text{worstCase}} = f_I + 4 \cdot f_I \cdot nr \quad [\text{s}^{-1}] \quad (2.20)$$

Where:

f_I	is the injection frequency of a box	$[\text{s}^{-1}]$
nr	is the quantity of relay nodes in a box	[1]
ti_n	is the quantity of transmissions coming from box n	$[\text{s}^{-1}]$

The traffic intensity is however dependent on the packet delivery rates, and by extension of that, the traffic intensity itself. The transmission frequency of boxes 1 and 2 is approximated in Appendix A as Equations (2.21) and (2.22).

$$ti_1 = f_I \quad [\text{s}^{-1}] \quad (2.21)$$

$$ti_2 = f_I \left(q_i(ti_s)\, nr + q_i(ti_s)\, q_r(ti_s, nr) + 2\, nr\, q_{mp}(ti_1, ti_s, nr) + 1\right) \quad [\text{s}^{-1}] \quad (2.22)$$

Where:

q_{mp}	is Pr(an injection from either box 2 or 3 reaching relay nodes in box 2)	[1]
ti_s	is given by $ti_s = ti_1 + ti_2 + ti_3 = ti_2 + ti_3 + ti_4$	$[\text{s}^{-1}]$

The probability $q_{mp}(ti1, ti_s, nr)$ is given by Equation (2.23).

$$q_{mp}(ti_1, ti_s, nr) = q_i(ti_s) + q_i(ti_s) \cdot (1 - q_i(ti1)) \cdot q_r(ti_s, nr) \quad [1] \quad (2.23)$$

With the traffic intensities, injection probability, and relaying probability determined, the packet arrival probability from node A to node B can be determined.

Packet arrival probability: The probability of the packet from node A arriving at its destination is equal the probability of one successful injection and two consecutive successful relayings. The last relaying has a lower probability of colliding, due to the nodes in the right-most box only being adjacent to one box. The packet arrival probability is written in Equation (2.24).

$$Pr\,(\text{packet arrival}) = q_a = q_i(ti_{1+2+3}) \cdot q_r(ti_{2+3+4}, nr) \cdot q_r(ti_{3+4}, nr) \quad [1] \quad (2.24)$$

Where:

q_a	is $Pr\,(\text{packet arrival})$	[1]

The reason for $ti = ti_{1+2+3}$ for the injection and $ti = ri_{2+3+4}$ or $ti = ti_{3+4}$ for the relayings now follows. If the injection of a packet P from node A is to reach box 2, it must not collide with other transmissions within the range of box 2's nodes. This means that it must not collide with any transmissions from boxes 1, 2, and 3. The traffic intensity for the injection is therefore $ti = ti_1 + ti_2 + ti_3 = ti_{1+2+3}$. The same reasoning is used

for the relayings. If the packet P is to reach box 3, it must not collide with any other transmissions within range of the nodes in box 3. The traffic intensity for first relaying is therefore $ti = ti_2 + ti_3 + ti_4 = ti_{2+3+4}$. The traffic intensity for the last relaying follows the same reasoning.

The packet arrival probability q_a dependent on the number of relays per box nr and the injection frequency f_I for a sparse network is illustrated on Figure 2.14.

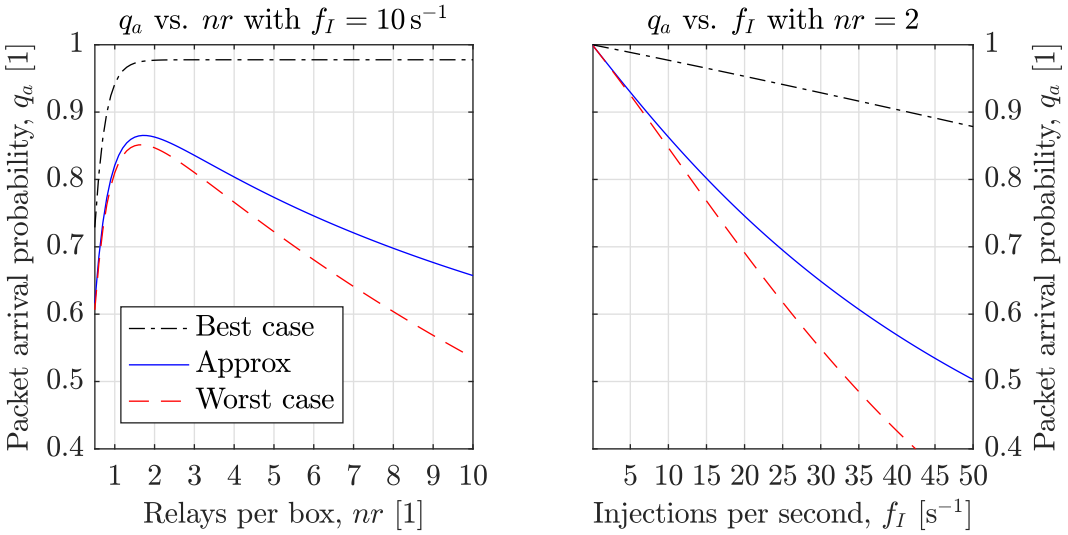


Figure 2.14: Plot of the packet arrival probability q_a in respect to the quantity of relays per box nr and the injection frequency f_I.

From Figure 2.14 it is seen that the number of relays in the boxes nr should be kept low to achieve a high packet arrival probability. The graph has a maximum at $nr \approx 1.7$, showing that the minimum amount of relay nodes is not necessarily the optimal choice.

2.6.3 Conclusion of Bluetooth Mesh use case analysis

From this section it is seen that it is feasible for a BM mesh network to work with the BM protocol stack under the assumptions made. However with complex 1000+ node networks, high packet loss is to be expected, unless counter measures are implemented.

When distributing relays through a BM network it is seen that the amount of relays should be kept low. Relay roles should therefore be assigned carefully to get a good network. This analysis also shows that it might not be the best solution to have the lowest possible quantity of relays connecting the network. Assigning more relay roles up till a certain point should increase the packet arrival probability. In sparse networks, the packet arrival probability might be increased by a few percent by repeating injections based on Figure 2.12. Injections should however not be repeated in dense networks, as it should result in a significantly reduced packet arrival probability.

2.7 Conclusion on Bluetooth Mesh analysis

Bluetooth Mesh (BM) is a standard with device requirements and a protocol stack targeted for small low powered devices such as Internet of Things (IoT). BM aims to serve as a network for simple and cheap devices, and therefore prioritises simplicity over performance.

BM follows the mesh network topology and introduces four different roles: ordinary, relay, low power, and friend. Low power nodes form "friendships" with friend nodes who caches packets destined for their befriended low power nodes. The friend nodes relays the cached packets when the low power nodes powers on. Relay nodes relays any packets on the network with correct authorization codes and through this connects the whole mesh network. The distribution of relay nodes greatly affects the performance of the network. There must be enough relay nodes to connect all nodes in the network. Too many relay nodes increases the risk of collisions, reducing the performance of the network. The standard does not specify how relay roles should be distributed throughout the network.

In Section 2.3 it was found that physical transmissions happens on three different channels spread across the Bluetooth spectrum. These channels are called Advertisement (ADV) channels and are placed out of the way of the most commonly used Wi-Fi channels. All transmissions are modulated with Gaussian Frequency Shift Keying (GFSK).

When a packet is transmitted, it is transmitted on all three ADV channels sequentially. Nodes can choose to listen to one of the three ADV channels selected at random, to receive packets. The use of three different channels allows the devices to switch to another channel if one channel is found noisy. This adds redundancy to the system at the cost of having to transmit everything thrice and using a wider frequency band. Packets have a maximum size of 47 B and transmissions happen at $1\,\mathrm{Mb\,s^{-1}}$.

In Section 2.4 it was found that the link layer implements a simple Medium Access Control (MAC) scheme which is completely uncoordinated. There is no collision detection or acknowledgement/synchronisation packets. This means that if a collision happens, the packet is simply lost.

In Section 2.5 it was found that the network layer implements flooding with a Time To Live (TTL) counter and a network packet cache to avoid packets traversing the network indefinitely. Due to these choices, there is a relatively high probability of losing packets. Flooding creates a risk of overflowing the network, causing many collisions, which in turn results in lost packet due to the absence of acknowledgements and the likes known from connection oriented systems. There is little to no overhead in the syste m, since there is no need for system wide synchronization or knowledge about the layout of the network. If the traffic intensity and packet sizes are low enough, the performance should not be compromised much relative to the increased simplicity of the devices and reduced communication overhead.

In Section 2.6 a Bluetooth Mesh use case analysis was made. With a dense network without relays, where all nodes are within range the packet arrival probability is 0.47 for a 1000 node network. Increasing the quantity of nodes, decreases the packet arrival probability further. The packet arrival probability can be increased by decreasing the traffic intensity or packet length.

In a case where a node is not in range of the whole network, relays are necessary. It was found that the quantity of relay nodes should be kept low. The analysis however also shows that the minimum number of relay nodes connecting the network is not the optimal relay role distribution. Relay role assignment should therefore be done in such a way that the number of relay nodes is kept low, while maintaining a bit of redundancy.

In sparse networks it might be beneficial to the packet arrival probability to introduce repetitions of injections, even though this increases the traffic intensity and probability of collisions. Repetitions of injections should however be avoided in dense networks.

Problem statement 3

The lower layers of the Bluetooth Mesh (BM) stack was analysed in Chapter 2. The analysis showed that the network is prone to packet loss due to the chosen Medium Access Control (MAC) and Network Layer (NL) protocols. Nodes simply transmit packets whenever new data is ready without listening to the channel beforehand. The transmitter does not attempt to detect collisions and therefore has no way of knowing whether the packet was successfully received. The NL implements flooding which has a risk of packet loss. The Physical Layer (PL) also leaves room for improvement due to the loose requirements, on for example the modulation index of the chosen Gaussian Frequency Shift Keying (GFSK). These choices are however made as a compromise between simplicity and reliability, under the premise that a network with short and few packets does not benefit much from the increased complexity.

Changes to the BM stack should improve the performance without compromising on simplicity, since this has a high priority in the BM standard. This makes connection-oriented communication, smarter routing schemes, stricter requirements for the PL, a more complicated MAC, and quality of service such as prioritized packets invalid solutions.

Dynamic repetition of packets and relay behaviour, quantity, and distribution could be altered without introducing more complexity on devices. A message queue systems could be implemented on relay nodes to reduce probability of packet loss due to multiple packets traversing the network. The length of a random wait between packet receipt and relaying could be varied depending on factors such as traffic intensity. Ordinary nodes and relay nodes could repeat transmissions based on traffic intensity to optimize Packet Delivery Ratio (PDR). Lastly, relay node quantity and distribution could be optimized to favour network performance.

Since the BM does not state how relay nodes should be distributed, there is a risk of nodes begin left unconnected from the rest of the network due to absence of relay nodes. Section 2.6.2 shows the importance of optimized relay placement in respect to network performance, demonstrating that simply over-provisioning relay roles, to connect the network, will result in performance degradation. Due to these observations, the remainder of the project will be focused on answering the following problem statement:

How can relay functionality in a Bluetooth Mesh network be assigned favouring performance, and how can this assignment be done automatically?

With the problem statement created, a solution proposal will be described in the next chapter.

Solution proposal & performance metrics 4

This chapter proposes a solution to the problem statement from Chapter 3 and sets requirements for a realisation. This chapter also defines a range of performance metrics which are to be used to evaluate the performance of different realisations.

4.1 Solution proposal

The analysis (Section 2.6.2) shows that the packet arrival rate is improved by reducing the quantity of relay nodes within the network. Relay nodes can however not simply be removed since they are essential for connecting the network. The analysis shows that having a few redundant relay nodes as opposed to the absolute minimum quantity of relay nodes connecting the network results in improved packet arrival probability.

To optimize the performance of Bluetooth Mesh (BM), relays should connect the whole network while remaining low in quantity. The network should contain a few redundant nodes enabling multipathing.

To distribute relay nodes in the network different types of solutions can be considered. Relay functionality could be distributed completely manually by a user of the network. Relay functionality could be distributed by a dedicated device with the sole purpose of monitoring the network and assigning relay functionality. Lastly the task of distributing relay functionality could be assigned to the nodes on the network.

This project proposes that the network should configure itself and distribute relay roles automatically. This can be done in two ways: distributed where decisions are made based on information about the nodes within range, or central by a node who gathers information on the whole network.

With a distributed solution all nodes are participating in the network configuration process. Nodes will need to exchanges information about themselves with other nodes in their neighbourhood. Based solely on this shared information, nodes will decide if they should be a relay node, or in some cases who in their neighbourhood should be a relay node.

With a central solution, one privileged node has to collect information from all other nodes in the network, to gain knowledge about the network topology. From this information, the privileged node decides which nodes should be relay nodes.

Distributed and central solutions both have different advantages and disadvantages. These are now briefly discussed.

With a central solution one node has to be chosen as a "central node". Choosing which node should be the central node can be a complex problem in itself. The central node should have a certain amount of memory and computational power. Since it needs to store information about all nodes in the network and from this information calculate which set of nodes should be relay nodes.

A disadvantage of a central solution is that every single node needs to send information about itself to the central node. Coordinating how this information is sent can de difficult, since there are no relay roles assigned, and the topology of the network is unknown. To ensure that the information can reach the central node, the information about every node can be flooded by all nodes. This approach can however cause many packet collisions and most likely packet loss, unless a very long time is dedicated to information gathering. Packet loss will result in incomplete information, which can in worst case leave some nodes unconnected after the setup.

An advantage of a central solution is that with complete information about the network, a central node can theoretically determine the optimal set of relay nodes. The optimal set of relays might however not be possible to calculate due to limitations on computing power, time, and battery capacity. Sufficient approximations should be possible within acceptable time frames.

With a distributed solution, a node is only required to store information about its own neighbourhood, and based on this information decide if it should be a relay. With a distributed solution it is unlikely to find the optimal set of relay nodes, since only limited information is available to each node.

An advantage of the distributed solution is that when a node wants to exchange information with its neighbourhood,the node needs to flood the neighbourhood. This should result in a significantly lower risk of packet loss, and thereby incomplete information. Every node can also be made responsible for ensuring that they are connected to the rest of the network, removing the risk of unconnected nodes.

Based on the disadvantages of the two solution types it is chosen to focus primarily on a distributed solution for this project.

With a description of the desired solution given, requirements and performance metrics for the proposed solution are given.

4.2 Solution requirement

Only one requirement for the solution exists:

The solution shall guarantee that all nodes are connected.

If two nodes are not directly within range of each other, they should be able to communicate with each other through one or more relays.

Different realisations of distributed relay assignment will be evaluated based on the following performance metrics.

4.3 Performance metrics

In order to measure the performance of the proposed solution, several performance metrics are defined. In Table 4.1 the chosen performance metrics are listed. Arguments for using these specific metrics, and how they are calculated, are given in Sections 4.3.1 to 4.3.5.

Table 4.1: The performance metrics of the desired solution.

Number	Performance metric
1	Configuration time
2	Packet Delivery Ratio
3	Data throughput
4	Latency
5	Fairness

Most performance metrics describes the performance of the network after the configuration phase. Therefore these performance metric will be measured after the relay nodes have been distributed.

4.3.1 Configuration time

The configuration time is the time frame from when the network starts configuring itself, until the setup is done. If the algorithm is implemented in a distributed way, then the time to configure is given as the time from when the network starts, until all the nodes in the network have been configured.

4.3.2 Packet Delivery Ratio

The Packet Delivery Ratio (PDR) is how large a fraction of sent messages are delivered successfully. It is measured by sending messages from an arbitrary node to another arbitrary node. Selection of sender and receiver node should be chosen uniformly. Time between messages should also be chosen at random.

4.3.3 Data throughput

Data throughput is a measure of how much raw data that is sent through a network. Data throughput is measured in $\mathrm{b\,s^{-1}}$. It is calculated by taking all delivered bits and dividing with the time taken, given in Equation (4.1).

$$\text{Throughput} = \frac{\text{Total delivered bits}}{\text{Total time}} \qquad [\mathrm{b\,s^{-1}}] \quad (4.1)$$

4.3.4 Latency

Latency is how long it takes for a messages to travel from source to destination. This can either be measured as hops, or time in seconds and both will be done in this project.

4.3.5 Fairness

The fairness metric is a measure of how well a performance metric of the network is shared across nodes. To measure this, the equation made by Jain et al.[9] in Equation (4.2) is utilised.

$$J(x_1, x_2 \dots x_n) = \frac{\left(\sum_{i=1}^{n} x_i\right)^2}{n \cdot \sum_{i=1}^{n} x_i^2} \qquad [1] \quad (4.2)$$

Where:

J	Raj Jain's fairness equation	[1]
x_i	Performance metric of the i'th node	$[\mathrm{b\,s^{-1}}]$
n	number of nodes	[1]

This gives a number between 0 and 1, where 1 is the most fair and 0 is least fair.

Another way to measure fairness is to divide the element with the worst performance with the element that has the best performance. This is given by Equation (4.3)

$$\text{fairness} = \frac{x_{\text{worst}}}{x_{\text{best}}} \quad (4.3)$$

Where:

fairness	fairness	[1]
x_{worst}	Worst performance	[varies]
x_{best}	Best performance	[varies]

This gives another picture of fairness, since it compares the best and worst performances. Equation (4.3) can give a better picture of fairness, since it is a demand for the algorithm to connect all nodes.

As a demonstration, consider a scenario where 1 in a 1000 nodes is unconnected, and all others are perfectly connected.

$$J(x_1, x_2 \dots x_n) = \frac{999}{1000 \cdot 999} = 0.999 \quad (4.4)$$

$$\text{fairness} = \frac{0}{1} = 0 \quad (4.5)$$

It is clear that this network is not fair since one node is not connected. Equation (4.4) shows that the network is almost as fair as it can be. On the other hand, Equation (4.5) shows that the network is not fair at all.

When considering node fairness both Equation (4.2) and Equation (4.3) is used. Furthermore, only PDR is used as a variable in the equations.

4.4 Validation of solution

The performance metrics of a proposed solution can be measured in different ways: by calculations, simulations, or by testing a real world implementation.

Validation the network by testing a real world implementation, requires resources not available to this project. The required amount of nodes for a distributed Bluetooth Mesh (BM) network, is not available to the project. Evaluating the performance metrics of a proposed solution solely based on probability calculations is assessed to be outside the scope of this project.

Since neither validation by calculations or by real world implementation are feasible, it is chosen to validate the proposed solution with a network simulator. It is arbitrarily chosen to use the program MATLAB by MathWorks to make the network simulator. The network simulator is designed and described in Chapter 5.

Network simulator 5

It was chosen in Section 4.4 that the proposed solutions are to be validated by means of simulations. Since no known simulator exists, a simulator is developed from scratch. It is arbitrarily chosen to program the simulator in MATLAB by MathWorks.

The purpose of this chapter is to design and document the constructed network simulator.

Since this is a student project with limited time and resources, not all aspects of a complete network simulator is possible. It is therefore necessary to make some delimitations and estimations when designing the simulator. To simplify the construction of the simulator it is chosen to delimit the network simulator from:

- Three dimensional transmissions. Nodes transmit on a two dimensional grid.
- Dynamically changing networks. Only static networks will be considered, meaning that no node will move during the simulation.
- Noise due to Wi-Fi or other communication systems working in the same band(s). Only interference from other nodes will be considered.
- Layers above the network layer of the Bluetooth Mesh (BM) protocol stack.

As reasoned in Section 5.2.2, the channel will exclusivity be modelled as a simple Additive White Gaussian Noise (AWGN) channel. That effects such as power delay, Shadowing, and small scale fading are all omitted in this project. The purpose of the network simulator is to test the effect of different Network Layer (NL) behaviours.

The made assumptions will result in an inaccurate representation of reality. Not including external noise sources, dynamic network changes, shadowing, etc. will result in a different Packet Delivery Ratio (PDR) in the simulations compared to reality. The use of a simple AWGN channel might result in a offset in the found PDR compare to reality. It is assessed that this offset will have minimum impact in system performance and the simple channel model will be sufficient for the purpose of the project.

With the scope of the network simulator defined, the simulator will now be designed. The design is begun by presenting an overview of the simulators structure.

5.1 Overview of the simulator

The simulator has three protocol layers: the Physical Layer (PL), the Link Layer (LL) and the Network Layer (NL). The NL checks if a node is transmitting or receiving and if a received packed shall be relayed. The LL handles channel hopping and keeps a message queue. The PL calculates Signal to Interference plus Noise Ratio (SINR) and decides if a packet is correctly received or not.

In the simulator nodes exists as objects. The node object holds the information, values, and features of the node.

When the simulator is run, different input parameters can be used dependent on what kind of simulation is desired.

The simulator has a time resolution of 8 µs. This time resolution is chosen based on the time it takes a Bluetooth Mesh (BM) device to transmit one byte, $\Delta t = \frac{8\,\mathrm{b}}{1\,\mathrm{Mb\,s^{-1}}} = 8\,\mu\mathrm{s}$. This is a reasonable compromise between simulation length and ability to distinguish multiple transmitting nodes.

With an overview of the simulator given, a detailed description of the constructed simulator will now follow, starting with the physical layer of the simulation.

5.2 Physical Layer

The Physical Layer (PL) handles transmissions from one node to another and calculates if the packet is correctly received. This calculation takes multiple factors into consideration.

First, the signal strength of the received signal is calculated from a path loss estimation, calculated from the transmitter to the receiver. Then, the interference from other nodes are calculated and added together. From these calculations a Signal to Interference plus Noise Ratio (SINR) is calculated and used to estimate the probability of a packet error.

5.2.1 Estimating received signal strenght from a path loss model

The IEEE 802.15.4a channel modeling subgroup has made path loss models for residential and office areas [10]. As described in Section 2.3 Bluetooth operates in the 2.402-2.4835 GHz frequency band. Therefore a path loss model specified for a band of 2-10 GHz is used. This model is given by Equation (5.1).

$$\mathrm{L}(f,d) = 0.5 \cdot \mathrm{PL}_0 \cdot \eta_{\mathrm{Tx}} \cdot \eta_{\mathrm{Rx}} \cdot \frac{\left(\frac{f}{f_c}\right)^{-2(\kappa+1)}}{\left(\frac{d}{d_0}\right)^{n}} \tag{5.1}$$

Where:

$\mathrm{L}(f,d)$	is the loss of power	[1]
f	is the frequency	[GHz]
f_c	is the reference frequency of 5 GHz	[GHz]

d	is the distance	[m]
d_0	is the reference distance of 1 m	[m]
κ	is the frequency dependence of the path loss	[1]
PL_0	is the path loss at 1 m distance	[1]
n	is the path loss exponent	[1]
η_{Rx}	is the efficiency of receiver antenna	[1]
η_{Tx}	is the efficiency of transmitter antenna	[1]

Since the frequencies used for Bluetooth Mesh (BM) is approximately constant across channels, the path loss model is reduced to $\mathrm{L}(d)$ when $f = 2.45\,\mathrm{GHz}$ is assumed.

The parameters which should be used for Equation (5.1) is given in Table 5.1. Molisch et al. [10] specifies a set of parameters based on measurements. For residential environments the parameters are measured within the range 7-20 m and for office environment the measurements are in a range of 3-28 m.

Table 5.1: Parameters used for path loss modeling [10].

	Residential		**Office**	
	LOS	**NLOS**	**LOS**	**NLOS**
PL_0[dB]	43.9	48.7	35.4	59.9
n	1.79	4.58	1.63	3.07
κ	1.12 ± 0.12	1.53 ± 0.32	0.03	0.71
$\eta_{\mathrm{Tx/Rx}}$	0.5	0.5	0.5	0.5

The parameter $\eta_{\mathrm{Tx/Rx}}$[1] in Table 5.1 are taken as an average between goal and minimum acceptable from [11]. For the residential case κ is modelled as a uniform random variable with mean 1.12 and limits ± 0.12 for Line Of Sight (LOS) and mean 1.52 and limits for ± 0.32 Non Line Of Sight (NLOS).

The different path loss models behave as illustrated in Figure 5.1. It is seen from Figure 5.1 that the models for LOS results in a path loss that is less than the free space model, and for NLOS the path loss is much greater. This behaviour matches the reality where, if there is LOS, then there is also reflections that can amplify[2] the signal.

[1] η_{Tx} and η_{Rx} are set to the same value.

[2] These reflections are not necessarily in phase with the original signal, and can then act as a cancellation/noise signal. In office/residential areas, the distances are small, so those calculations have been omitted.

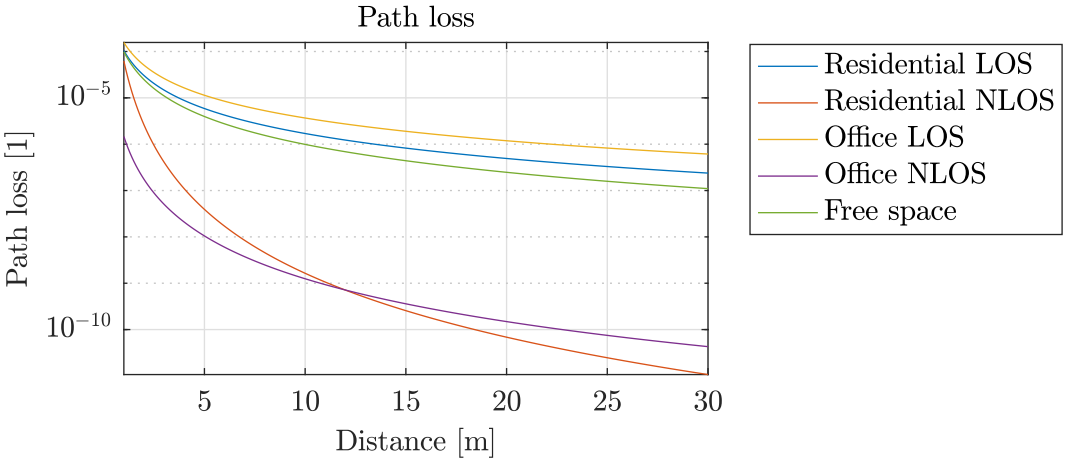


Figure 5.1: Different path loss models in relation to free space path loss.

This project exclusively uses the office LOS model, since the precise value of the path loss function is irrelevant for the purpose of the project. The important part of the path loss model is the overall trend of the path loss over distance, which is approximately the same for all models. A higher path loss results in a sparse network seen from a nodes point of view. Low path loss equals a dense network. The office LOS is chosen since it is symmetrical, in the sense that the path loss from a node u to a node v is the same as from v to u.

The received power is calculated by Equation (5.2) where j is the node with the strongest signal

$$P_{\mathrm{Rx},j} = P_{\mathrm{Tx},j} \cdot \mathrm{L}(d_j) \quad [\mathrm{W}] \quad (5.2)$$

Where:

$P_{\mathrm{Rx},j}$	is the power received from node j	[W]
$P_{\mathrm{Tx},j}$	is the power transmitted by node j	[W]
$\mathrm{L}(d_j)$	is the path loss from transmitter to receiver	[1]
d_j	is the distance transmitter node j to receiver	[m]
j	is the node with the strongest signal at the receivers position	[1]

5.2.2 Signal to Interference plus Noise Ratio and Block Error Probability

Signal to Noise Ratio (SNR) can be calculated, excluding interference, by Equation (5.3)

$$\mathrm{SNR} = \frac{P_{\mathrm{Rx},j}}{S_i} \quad [1] \quad (5.3)$$

Where:

SNR	is the signal to Noise Ratio	[1]

$P_{\mathrm{Rx},j}$	is the power received	[W]
S_i	is the receiver sensitivity	[W]

The interference is equal to the collective signals received that are not the strongest signal at the receiver's position. The interfering signals are added together and used to calculate the SINR.

From [12] it is seen that modelling of the physical domain should be done so that noise is additive. The interference P_i is therefore given by Equation (5.4). The interference is the sum of noise emitted from nodes that are transmitting on the same channel as the receiver is listenin. It is assumed that transmissions on other channels is narrow-banded enough to not interfere. The power received calculated in Equation (5.2) is subtracted since the strongest signal is not classified as noise.

$$P_i = \left(\sum_{n \in \mathrm{Tx} \cap \mathrm{Ch}}^{N} P_{\mathrm{Tx},n} \cdot \mathrm{L}(d_n) \right) - P_{\mathrm{Rx},j} \quad [\mathrm{W}] \quad (5.4)$$

Where:

P_i	is the power of interference	[1]
$P_{\mathrm{Tx},n}$	is the power transmitted by node n	[W]
$P_{\mathrm{Rx},j}$	is the power received from node j	[W]
$\mathrm{L}(d_n)$	is the path loss from node n to receiver	[1]
d_n	is the distance from node n to receiver	[m]
Tx	is the set of nodes currently transmitting	[1]
Ch	is the set of nodes on the channel of the receiver	[1]

The SINR is calculated by Equation (5.5), interference from other transmitters and sensitivity of the receiver is added together.

$$\mathrm{SINR} = \frac{P_{\mathrm{Rx},j}}{P_i + S_i} \quad [1] \quad (5.5)$$

Where:

SINR	is the signal to Interference plus Noise Ratio	[1]
P_i	is the power of interference	[W]
$P_{\mathrm{Rx},j}$	is the power received	[W]
S_i	is the receiver sensitivity	[W]

The Bit Error Rate (BER) is calculated using Equation (5.6) which is a model for a channel with Additive White Gaussian Noise (AWGN) from Section 2.3.1.

$$\mathrm{BER}_{\mathrm{AWGN}} = 0.5 \cdot \mathrm{erfc}\left(\sqrt{\alpha \cdot \mathrm{SINR}}\right) \quad [1] \quad (5.6)$$

Where:

SINR	is the Signal to Interference plus Noise Ratio	[1]
$\mathrm{BER}_{\mathrm{AWGN}}$	is the BER for a AWGN channel	[1]
α	is a constant parameter	[1]

Where erfc(x) is the complimentary error function given by Equation (5.7).

$$\text{erfc}(x) = 1 - \text{erf}(x) = \frac{2}{\sqrt{\pi}} \int_x^\infty \text{e}^{-t^2}\, \text{d}t \qquad [1] \quad (5.7)$$

Another channel used to model the BER is the fading channel. This model is given by Equation (5.8) from [8].

$$\text{BER}_{\text{fading}} \approx 0.5 \left(1 - \sqrt{\frac{1}{1 + \frac{1}{\text{SINR}}}} \right) \qquad [1] \quad (5.8)$$

Using the BER, the Block Error Probability (BLEP)[3] is derived for constant BER, this is done in Equation (5.9).

$$\begin{aligned} \text{BLEP} &= Pr(\text{one or more bits are wrong}) \\ \text{BLEP} &= 1 - Pr(\text{all bits are correct}) \\ \text{BLEP} &= 1 - (1 - Pr(\text{one bit is wrong}))^{\text{Bits sent}} \\ \text{BLEP}(\text{BER}, \text{n}) &= 1 - (1 - \text{BER})^n \end{aligned} \qquad [1] \quad (5.9)$$

Where:

BLEP	is the Block Error Probability	[1]
n	is the length of the message in bits	[1]
BER	is the calculated Bit Error Rate	[1]

For a nonconstant BER, the BLEP is calculated by taking the product of the probability that each individual bit is correct.

The distance at which a device can reliably transmit and receive depends on the parameter α in Equation (5.6). The BLEP is plotted in Figure 5.2 as a function of distance for different values α and for the fading channel.

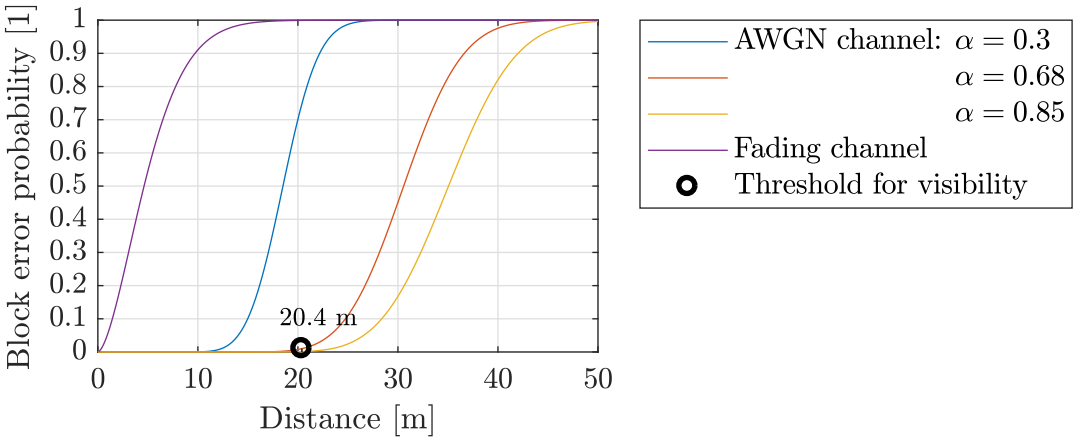


Figure 5.2: Plot of different channels and coefficients. The threshold for visibility is used in Chapter 6, and is defined as the range at which there is a BLEP of 1% assuming no collisions or interference.

[3]Where block denotes the length of the packet

It is seen how the BLEP worsens when either using a fading channel, or using a AWGN channel with low α-value. This makes sense regarding a low α-value, since from [8], it is seen that a low α-value equal low use of frequency space. The choice of channel model only affects the reception range of the nodes, and therefore simply results in a scaling of the distance.

Since the focus of this project is on the distribution of relays, the absolute value of the reception range is insignificant as long as the chosen model follows a somewhat correct pattern. For this reason the AWGN channel is arbitrarily chosen.

The AWGN channel with an α-value of 0.68 is chosen, since this is given by [8, page 1049] as a value for Gaussian filtered Minimum Shift Keying (GMSK) with a bandwidth bit period product $BT = 0.25$, which should result in a BER close to the BM GMSK with $BT = 0.5$.

A message is then received if Equation (5.10) is true

$$\text{BLEP}(\text{BER}, n) \leq X \tag{5.10}$$

Where:

X	is a uniform random variable between 0 and 1	[1]

The size of the simulated area and node density is determined based on Section 2.6.

5.2.3 Area

The size of the simulated area is based on the office space in 2.6. In that office space, there are 158 nodes that are relays. The office space is 27.4 m by 20.45 m which gives an area of $560.33\,\text{m}^2$. Nodes per square meters are given by Equation (5.11). In Equation (5.12), the length is calculated for the scenario with 1000 nodes

$$\text{nodesPerSquareMeter} = \frac{158}{560.33\,\text{m}^2} = 0.282\,\text{m}^{-2} \tag{5.11}$$

$$\text{Length}_{1000} = \sqrt{\frac{1000}{\text{nodesPerSquareMeters}}} = 59.552\,\text{m} \approx 60\,\text{m} \tag{5.12}$$

5.3 Link Layer

This section describes how the Link Layer (LL) is implemented in the simulator, as well as some implementation specific choices that have been made in regards to node behaviour. This section will describe the implementation of:

- LL as state machine
- Scanning and transmission windows
- Message queue
- Medium Access Control (MAC)

The LL controls the behaviour of the Physical Layer (PL). Section 2.4 describes how the LL of the Bluetooth Mesh (BM) protocol stack is implemented as a state machine. In the simulator the LL is also implemented as a state machine. In the simulator the state machine holds a number of global values, these variables are used in the PL to decide upon a behaviour. This way the behaviour of the PL is controlled by the LL by changing these global variables.

A list of the different variables and their meaning, are summarised in Table 5.2

Table 5.2: Variables that is used by the LL

Name	Range of values	Purpose
`currentlyTx`	Boolean	Node is physically transmitting message
`currentlyRx`	Boolean	Node is scanning
`channel`	1-3	Channel scanning or transmitting
`receiving`	Boolean	Node is receiving a message
`sending`	Boolean	Node is sending a message (across random delays)
`channelChangeWait`	Any positive integer	Random delay between channel change
`relayWait`	Any positive integer	Random delay after decision to relay
`messageSize`	Any positive integer	Size of the receiving or transmitting message
`messageCounter`	Any positive integer	How many bytes of the message has been received or transmitted

The behaviour of the LL is described by the pseudocode in Algorithm 1.

Algorithm 1 Link Layer behaviour

```
1: if relayWait = 0 then
2:     currentlyTx ← 1
3:     currentlyRx ← 0
4:     channel ← 1
5: end if
6: if channelChangeWait = 0 then
7:     currentlyTx ← 1
8:     currentlyRx ← 0
9: end if
10: decrement relayWait
11: decrement channelChangeWait
12: if currentlyTx ∧ sending then
13:     if Message not sent fully then
14:         Continue sending
15:     else if Message fully sent then
16:         if channel = 1 then
17:             messageCounter ← 0
18:             channel ← 2
19:             currentlyTx ← 0
20:             channelChangeWait ← random number
```

```
21:       else if channel = 2 then
22:         messageCounter ← 0
23:         channel ← 3
24:         currentlyTx ← 0
25:         channelChangeWait ← random number
26:       else
27:         channel ← random number between 1 and 3
28:         currentlyRx ← 1
29:         currentlyTx ← 0
30:       end if
31:     end if
32: end if
33: if currentlyRx ∧ receiving then
34:     if Message not received fully then
35:       Continue receiving
36:     else if Message received fully then
37:       Send message to Network Layer
38:     end if
39: end if
```

If a node needs to relay a received packet, it runs the code in algorithm 1 with the counter `relayWait` set to an integer larger than or equal to 1, and `channelChangeWait` to less than 0.

The delay `channelChangeWait` is used for making the simulated bluetooth radio wait a random time before it can transmit again when it has changed to a channel. This value has been arbitrarily chosen to be maximally 10 ms.

From Algorithm 1 it is seen how the packet is transmitted on the three channels by setting `currentlyTx` = 1. Furthermore it is seen from the code how a message is received by setting `currentlyRx` = 1. The variables `sending` and `receiving` handles the message queue system, see Section 5.3.2.

It is chosen to implement the link layer so that it will not frequency hop while receiving a message. Since there is no external interference, it only frequency hops when sending messages. When it is done sending a message, it goes to a random channel.

5.3.1 Scanning and transmission windows

The way nodes schedule their scanning windows and transmissions is not dictated by the standard, and is therefore implementation specific. This project has chosen to implement scanning windows and transmissions scheduling in the following way.

A node will stay on the same channel and keep scanning until its next transmission is scheduled. After a transmission the node will choose another channel at random and continue scanning until its next transmission is scheduled. If a node is a relay and receives

a packet, if all conditions are met, the node will schedule the relaying. The scheduling is randomly in a pre-determined interval after receipt.

When a packet is received and it does not need to relay, the node returns to scanning on a the same channel as it received the messages on. When a node is about to begin a transmission it will drop out of the scanning state and transmit the packet on all three channels sequentially, before returning to scanning on a random channel.

5.3.2 Message Queue

The Bluetooth specifications does not mention a message queue, however nothing is preventing device vendors from implementing a message queue.

For the optimal network performance, a node should have a scanning duty cycle as close to 100% as possible [2, page. 47]. Whenever the node is not scanning the channel, potential packets might be lost. Therefore it would be best if the node scanned the channel while waiting on the `relayWait` to run out. However if a packet is received within this "extra" scanning interval the node needs to be able to store the message in a message queue, else nothing would have been gained from the additional scanning.

A message queue can also be used to implement simple congestion control mechanics. If a node sees that the queue is filling up, it can do multiple things. Such as drop messages, either in some suitable way or random. It could also send congestion messages to its neighbours in an effort to decrease traffic.

For the simulator it is decided to implement a simple message queue. The length of the message queue can be set as an option. If the message queue gets filled the oldest packet in the queue is dropped.

When there are packets in the queue, the node only schedules the relaying of the next packet. This means that the time it takes to get through the queue is dependant on the length of the queue, as opposed to a caching system where the relaying of a packet is scheduled independently of the rest of the traffic. The chosen queue implementation works in the same manner as a leaky bucket algorithm and has a higher risk of losing packets due to queue overflow. This can result in higher latency, but since the queue has built in congestion control it spreads the number of packets relayed per second evenly.

5.3.3 Medium Access Control

Since the BM standard does not specify any MAC, it is implementation specific. A node can break receipt to begin its own transmissions or change channel while it has successfully detected an incoming transmission and is receiving. This behaviour will however result in a worse packet delivery ratio as opposed to when nodes can not break receipt, as described in Section 5.6.2.

It is chosen to implement the MAC such that if a node receives the first byte of a message correctly, the node will continue listening for the full duration of the packet transmission.

The node will listen for the full duration even if one of the following bytes are corrupted. This means that from the first byte the node can:

- Detect the packet
- Read length of the packet [4]

If a node is listening on a channel and the node successfully receives the first byte of a message, the node will continue listening and try to receive the whole message before starting to transmit anything itself. Whenever a node changes channel it will listen to the channel before starting to transmit anything on the channel.

On the other hand, if a node changes to a channel that is occupied by a transmission, the node will not receive the first byte of the packet. If the first byte is not received the node is unable to detect the transmission and can therefore initialize a transmission of its own, causing a collision.

5.4 Network Layer

The Network Layer (NL) performs as is described by Figure 2.8. The simulator does not check the security check or group address. This is because it is out of scope of the project.

5.5 Simulation Parameters

When simulating the network, different parameters is changed, depending on what needs to be simulated. These parameters are listed and explained in Appendix C, where default values are given.

5.6 Network simulator validation

In order to validate the network simulator three simple scenarios are simulated. Test 1 will test the path loss and block error models. Test 2 will test relaying and test 3 will test collisions. The simulated results are compared to analytical expressions.

5.6.1 Test 1

In the first test the implementation of the chosen path loss and block error models are tested.

For this test a simple two-node network as seen on Figure 5.3 is considered.

[4] This is not specified in the standard but is chosen for making the simulation simpler

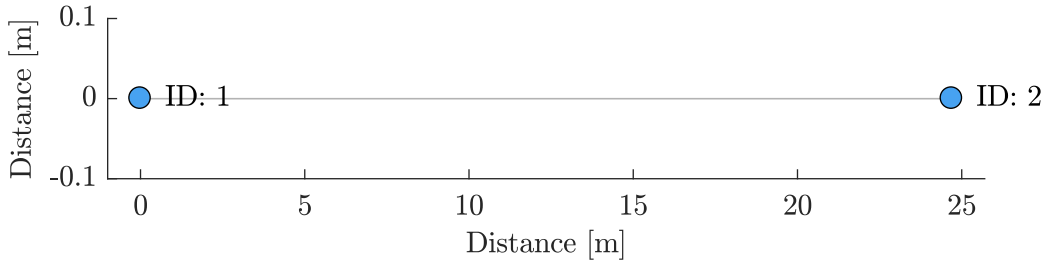


Figure 5.3: A two-node setup used for test 1. Node 1 transmits packets to node 2.

The Block Error Probability (BLEP) of a transmission from node 1 to node 2 can be calculated with Equation (5.9). At a distance of 24.8 m, it is seen from Figure 5.2 that the BLEP will be 0.1. In this two-node case the average Packet Delivery Ratio (PDR) is equal to $1 - \text{BLEP}$. This means a PDR of 0.9 is expected from the simulator.

The two-node network from Figure 5.3 is simulated in the network simulator, at 40 different traffic intensity levels with 1000 packet each, varying from $10 - 800$ packet/s. The simulation shows, with a 99.5% confidence interval, that the average PDR is 0.902 ± 0.005.

Thus it is seen that the resulting BLEP from the simulator matches the expected value from the calculations. Therefore the test is considered passed.

5.6.2 Test 2

For the second test the behaviour of a relay node is tested. In this test a three-node network as seen on Figure 5.4 is considered. Node 2 is a relay node connecting nodes 1 and 3. Node 1 transmits packets destined for node 3. Nodes 2 and 3 does not inject packets.

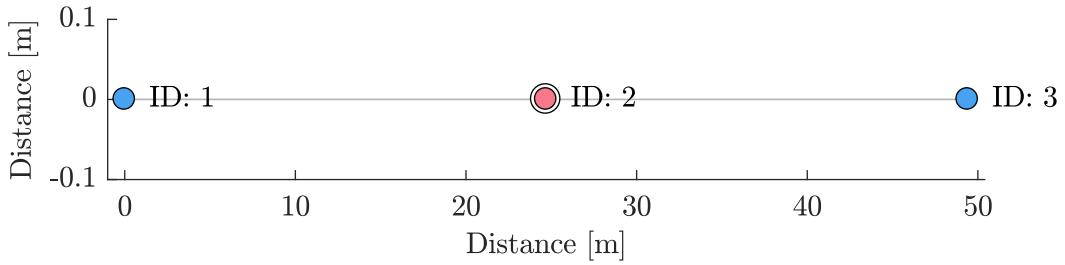


Figure 5.4: A three-node setup used for test 2. Node 1 transmits packets to node 3 through node 2.

Node 1 schedules all its packets before the simulation starts. The beginning of every transmission is scheduled randomly, from a uniform distribution, at a simulation tick within the interval $I = \left[47 \cdot 3 + 4, 2 \cdot \frac{\text{tick per second}}{\text{packets per second}} - (47 \cdot 3 + 4)\right] + t_{\text{previousTransmission}}$. The interval is chosen like this since a node is occupied $3 \cdot 47$ tick for transmitting on all three channel, as well as up to four ticks for channel switching, after beginning a transmission. This results in an average spacing between the transmissions of $t_{\text{avgSpacing}} =$

$\frac{\text{tick per second}}{\text{packets per second}}$. Code snippet 5.1 shows how the packets are scheduled, saved in the variable `msgTiming`.

Code Snippet 5.1: Matlab code for packet scheduling

```
1 msgTiming =0;
2 for temp = 1:(msgNum)
3    msgTiming(end+1) =randi([47*3+4,round( 2*(totalTime/msgNum)- (47*3+4) ) ]) +
         msgTiming(end);
4 end
```

To limit the possible ways packet loss can happen, the relay is set to always begin relaying the following tick after receipt. This way collisions and packet loss can only happen due to low power levels or if node 1 transmits packets in rapid succession.

From the node setup seen on Figure 5.4 it can be derived that the probability of a packet successfully being transmitted from node 1 to node 3 can be expressed as seen in Equation (5.13).

$$q_a = Pr(N2_{Rx}|N1_{Tx}) \cdot Pr(N3_{Rx}|N2_{Tx}) \tag{5.13}$$

Where:

q_a	is the packet arrival probability	[1]
$Pr(N2_{Rx}\|N1_{Tx})$	is the probability that node 2 successfully receives given that node 1 is transmitting	[1]
$Pr(N3_{Rx}\|N2_{Tx})$	is the probability that node 3 successfully receives given that node 2 is transmitting	[1]

Since node 1 is not in range of node 3, the probability that node 3 successfully receives a packet send by node 2 is equal to the probability of no bit errors when there are no collisions. Thus $Pr(N3_{Rx}|N2_{Tx}) = 1 - \text{BLEP} = 0.9$, since BLEP is given as 0.1 at a range of 24.8 m in Section 5.6.1.

As described in Section 5.3.3 nodes are implemented so that they will not start transmitting, if they have registered an incoming message. This is something which is implementation specific. Therefore both the case with and without this non-receipt breaking behaviour is considered in this test. The two cases differ only in the probability that node 2 successfully receives a packet send by node 1.

With the non-receipt breaking behaviour Appendix B derives an upper bound for the probability that node 2 successfully receives a packet send by node 1 which is given by Equation (5.14).

$$Pr(N2_{Rx}|N1_{Tx}) \leq q = \frac{1.8 \cdot k - n \cdot \left(261 + (2 \cdot tx + 1) \cdot \frac{q}{3}\right)}{2 \cdot k - 290 \cdot n} \tag{5.14}$$

Where:

tx	is the length of a packet in simulator ticks	[tick]
k	is the quantity of simulator tick in a second	[tick /s]
n	is the packet injections per second	[packet /s]

With the receipt breaking behaviour The probability that a packet from node 1 reaches node 2, if nodes can break receipt is also derived in Appendix B and is given by Equation (5.15). Since nodes have been implemented so that they can not break receipt, this probability serves as a lower bound for the probability $Pr(N2_{Rx}|N1_{Tx})$.

$$Pr(N2_{Rx}|N1_{Tx}) \geq q = \frac{1.8 \cdot k - n \cdot (261 + tx \cdot q)}{2 \cdot k - 290 \cdot n} \tag{5.15}$$

Where:

tx	is the length of a packet in simulator ticks	[tick]
k	is the quantity of simulator tick in a second	[tick /s]
n	is the packet injections per second	[packet /s]

With the expected packet arrival probability bounded by Equations (5.14) and (5.15), the network is simulated.

Simulation and results The variables seen in Table 5.3 are used for both the simulator and the calculations. On Figure 5.5 the simulation results and Equations (5.14) and (5.15) are illustrated.

Table 5.3: Table specifying the variables used for test 2.

Variable	Value	Unit
Packet injections per second n	$n \in [10, 650]$	packet/s
Packet length tx	47	tick/packet
Total packets m	5000	packet
Simulation ticks per second k	125 000	tick/s
Simulation total time	$\frac{m}{n} \cdot k$	tick
Random number generator seed	1	1

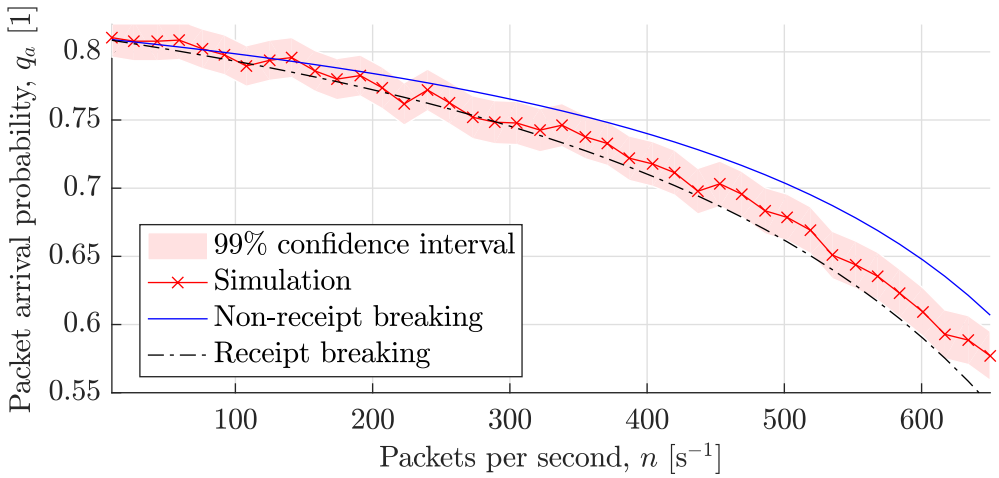


Figure 5.5: The results of the second test. The shaded area is the 99% confidence interval of the test result.

From Figure 5.5 it is seen that the simulation results lies within the area bounded by Equations (5.14) and (5.15), the test is therefore considered passed. Figure 5.5 also shows that the network benefits from nodes having a non packet breaking behaviour implemented.

5.6.3 Test 3

In the third test the packet collision behaviour of two nodes is tested.

In this test a simple three node setup as seen on Figure 5.6 is simulated. Node 1 is transmitting to node 3. Node 2 always transmits colliding packets at the exact same time as node 1 is transmitting. The distance d between nodes 1 and 2 is varied.

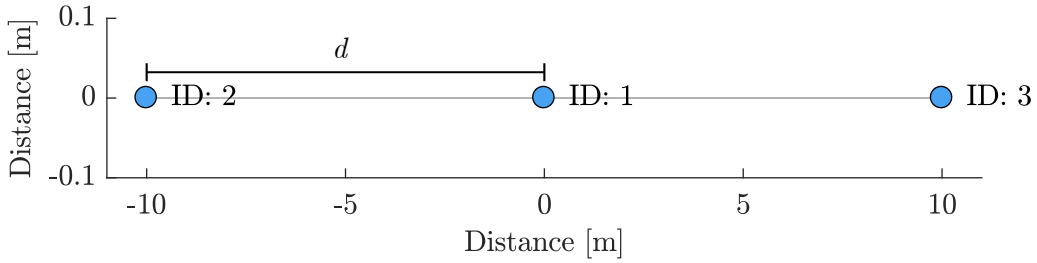


Figure 5.6: An illustration of the three node setup used for test 3.

For this test only the non-receipt breaking behaviour will be considered. If nodes 1 and 3 always transmits simultaneously, the probability of a packet successfully being transmitted from node 1 to node 3 can be expressed by using Equation (5.9) as done in Equation (5.16).

$$Pr(N3_{Rx}|N1_{Tx}) = q_a = 1 - \text{BLEP}(\text{BER}(d), 8 \cdot 47) = (1 - \text{BER}(d))^{8 \cdot 47} \tag{5.16}$$

Where:

d	is the distance between nodes 1 and 2	[m]
q_a	is the packet arrival probability	[1]
$Pr(N3_{Rx}\|N1_{Tx})$	is the probability that node 3 successfully receives given that node 1 is transmitting	[1]

Using the chosen Additive White Gaussian Noise (AWGN) channel, the Bit Error Rate (BER) is given by Equation (5.6) as Equation (5.17).

$$\text{BER}(d) = \text{BER}_{\text{AWGN}}(d) = 0.5 \cdot \text{erfc}\left(\sqrt{0.68 \cdot \text{SINR}(d)}\right) \tag{5.17}$$

Lastly, the Signal to Interference plus Noise Ratio (SINR) is calculated using Equation (5.5), dependent on the distance d, is given by Equation (5.18).

$$\text{SINR}(d) = \frac{\text{L}(10)}{\text{L}(10 + d) + 10^{\frac{-70}{10}}} \tag{5.18}$$

The pathloss $\text{L}(d)$ is given by Equation (5.1) with the parameters for office Line Of Sight (LOS) from Table 5.1.

With the probability of packet arrival q_a determined, the network is simulated. The results from the simulation together with the analytically derived packet arrival probability is

Table 5.4: Table specifying the variables used for test 3.

Variable	**Value**	**unit**
Distance between nodes 1 and 2 d	$d \in [10, 50]$	m
Packets per second n	600	packet/s
Packet length	47	tick/packet
Total packets m	5000	packet
Simulation ticks per second k	125 000	tick/s
Simulation total time	$\frac{m}{n} \cdot k$	tick
Random number generator seed	1	1

illustrated on Figure 5.7. The variables seen in Table 5.4, is used for both the simulator and the calculations.

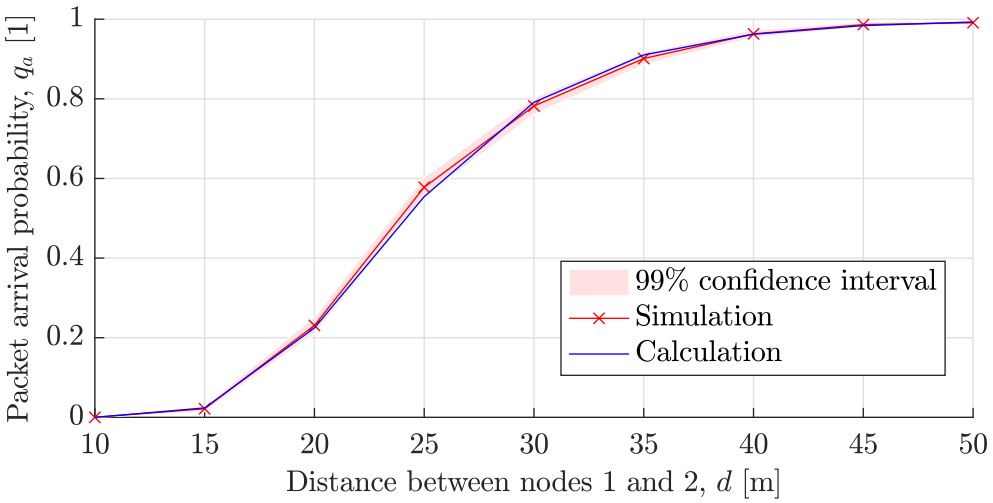


Figure 5.7: Graph showing the results from the simulation together with the analytically derived packet arrival probability. The shaded area is the 99% confidence interval of the test result.

Figure 5.7 shows that the expected packet arrival probability is within the 99% confidence interval of the simulated results.

With the found results, it is assessed that the results validates the simulator and the test is therefore accepted.

All the conducted tests have shown the network simulator to function as expected. Therefore it is concluded that the constructed network simulator works as intended.

Algorithm analysis 6

With a MATLAB network simulator designed and implemented as described in Chapter 5, it is possible to implement and simulate different relay distribution algorithms. The algorithms are run on different Bluetooth Mesh (BM) networks and their performance metrics, see Section 4.3, are compared after setup. Different relay distribution algorithms is described, implemented, and tested in the network simulator throughout this chapter.

Network operation can be divided into two states: setup and normal operation. Setup includes node discovery and connection while normal operation contains maintenance and ordinary usage. This analysis wishes to compare different algorithms based on their normal operation performance. To avoid implementation specific considerations and trade-offs having effects on the resulting set of relays, some assumptions are made during the setup phase:

- Communication between nodes is instantaneous.
- Communication between nodes is both without errors and collisions.
- All nodes have complete knowledge of their neighbourhood at any given time.

After the setup phase the network is tested without these assumptions so that collisions, etc. can happen.

When the different algorithm are tested in the network simulator the exact same network is used for all simulations to ensure a fair comparison. A network with 1000 nodes is uniformly generated, using the `rand` function in MATLAB. The generated network is seen on Figure 6.1.

The network seen on Figure 6.1 is used for all illustrations of the algorithms in this chapter. A node is said to be in range of another if there is less or equal to 1% probability of packet loss between them assuming no interference. With this definition, the average node is within range of 255.3 other nodes.

Before the relay distribution algorithms are described, some notations and definitions are defined.

All nodes in the network are denoted as the set G. A single node in G is denoted as u. The set of neighbouring nodes of a node u is denoted $N(u)$. One specific node in $N(u)$ is denoted as v.

A dominating set is denoted D and defined as a set $D \subset G$ where $D \cup N(D) = G$. A Connected Dominating Set (CDS) in G is denoted C. A set of nodes is a CDS if the set is both a dominating set, $C \cup N(C) = G$, and if all nodes in G is able to reach all other nodes in G by a path through nodes exclusively in C. [13, 14]

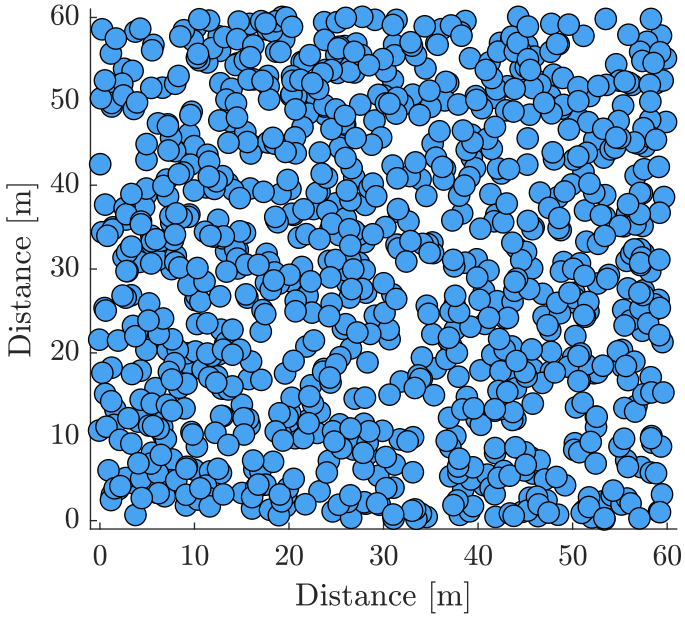
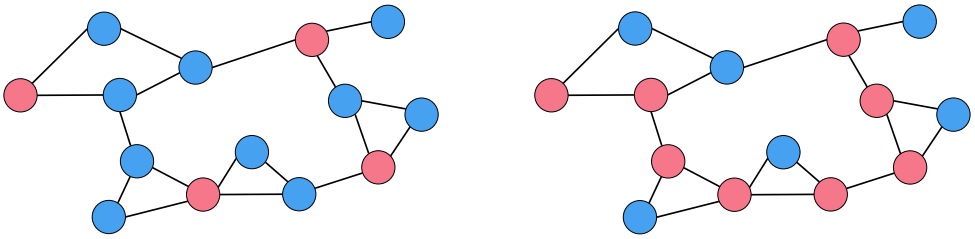


Figure 6.1: A 1000 node network topology.

On Figure 6.2 an example of a dominating set and a CDS is seen.



(a) A dominating set or nodes in a network. ***(b)*** A CDS of nodes in a network.

Figure 6.2: Illustation of two sets. The nodes included in the sets are marked with red. Figure 6.2a is a dominating set and Figure 6.2b is a CDS.

In the following sections different algorithms for constructing a CDS is presented where the first to be presented is a modified greedy algorithm which favours highest connectivity.

6.1 Highest connectivity based greedy algorithm (HighConnect)

A minimised dominating set can be constructed by using a greedy algorithm favouring nodes with highest connectivity. The minimised dominating set can afterwards be connected to form a Minimised Connected Dominating Set (MCDS) as done by Ghnimi et al. [15].

It is assessed that the size of the minimised dominating set might be reduced if relay assignment is done by a central node with knowledge of the whole network, as opposed to a distributed solution where nodes make decisions based on knowledge of their neighbourhood. Both distributed and central algorithm implementations are therefore simulated to see the different potentials.

6.1.1 Description of algorithm

Since the algorithm is designed as a distributed algorithm [15], it is initially described as such. The minor differences between the distributed and central implementations are given afterwards.

Ghnimi et al. [15] introduces three sets of nodes: dominating D, dominated $N(D)$, and undiscovered $U = G - \big(N(D) + D\big)$. Before network configuration starts, all nodes are undiscovered.

In the Bluetooth Mesh (BM) network topology dominating nodes are equivalent to relay nodes. The dominated nodes are equivalent of regular nodes and the undiscovered nodes are nodes unconnected from the rest of the network.

The concept of cluster identifiers are also introduced. Dominating nodes carries a cluster ID which is synchronised between all connected dominating nodes, thus enabling the individual node to determine whether the neighbouring dominating nodes are connected [15].

The algorithm is split into two phases: a construction phase and a connection phase.

In both phases of the algorithm, three rules apply:

1. When a node u joins the dominating set, it must inform all non-dominating neighbours $v \in \big(N(u) - D\big)$, to change their state to dominated. This way all neighbouring nodes of dominating nodes are in the dominated set $N(D)$.
2. When a node u joins the dominating set, it must synchronise its cluster ID with its neighbouring dominating nodes $v \in \big(N(u) \cap D\big)$.
3. When the cluster ID of a dominating node u is changed, it must synchronise its cluster ID with its neighbouring dominating nodes $v \in \big(N(u) \cap D\big)$.

A node u's cluster ID is synchronised by comparing the cluster ID of the node itself with the cluster IDs of its neighbouring dominating nodes $v \in \big(N(u) \cap D\big)$. The node u then assigns the cluster ID of itself and all its neighbouring dominating nodes to the highest

observed cluster ID. This operation together with rules 2 and 3 causes cluster IDs to propagate throughout a Connected Dominating Set (CDS) ensuring that all dominating nodes within a CDS has the same cluster ID.

The **construction phase** creates a minimised dominating set. This is done by allowing undiscovered node to join the dominating set, if they have a higher undiscovered neighbour count than all their neighbours. If there is a node $v \in N(u)$ exists such that $\left|N(u) \cap U\right| > \left|N(v) \cap U\right|$, then u will inform v that it must join the dominating set. If no such v exists node u will join the dominating set.

The construction phase runs until all nodes are discovered $U = \emptyset$. If there are any nodes v where $N(v) = \emptyset$, the construction phase can not be finished.

When the construction phase is complete a dominating set of nodes has been found. Nodes in the network will then advance to the connection phase.

The **connection phase** connects the minimised dominating set, creating a CDS. In this phase, dominated nodes runs two checks to determine whether they should join the dominating set or not.

1. If a dominated node u has two or more neighbouring dominating nodes $v \in \big(N(u) \cap D\big)$ with different cluster IDs, the node u must join the dominating set and synchronise cluster IDs.
2. If a dominated node u has a neighbouring dominated node $v \in \big(N(u) - D\big)$. Where the cluster ID of a node $x \in \big(N(u) \cap D\big)$ is different from the cluster ID of a node $y \in \big(N(v) \cap D\big)$. Then node u and v must join the dominating set and synchronise cluster IDs.

To set up the network, the distributed algorithm needs to run at least once on every node in the network. The network will not change after running the construction and connection phases.

The central implementation of the algorithm works in a very similar fashion.

In the construction phase of the central algorithm, a node u with the highest undiscovered neighbour count in the whole network joins the dominating set. This continues until all nodes are discovered $U = \emptyset$.

The connection phase of the central algorithm is split into two. Since cluster IDs are not propagated, the central node keeps track of which nodes to connect, to make the CDS. Since the central algorithm has knowledge about the whole network, the minimised dominating set has the potential to be smaller than what is created by the distributed implementation.

A fundamental differences between the distributed and the central is the needed information. The distributed algorithm only needs information about its neighbours and neighbours of neighbours. The central algorithm needs a list of all nodes and their neighbours.

6.1.2 Simulation of algorithm

With the algorithms described a MATLAB implementation of the algorithms have been made. The algorithms is seen in Code snippet J.1 and Code snippet K.1.

Assigning relay roles in the network illustrated on Figure 6.1 with the distributed highest connectivity based greedy algorithm yields 11 relay nodes, as shown on Figure 6.3.

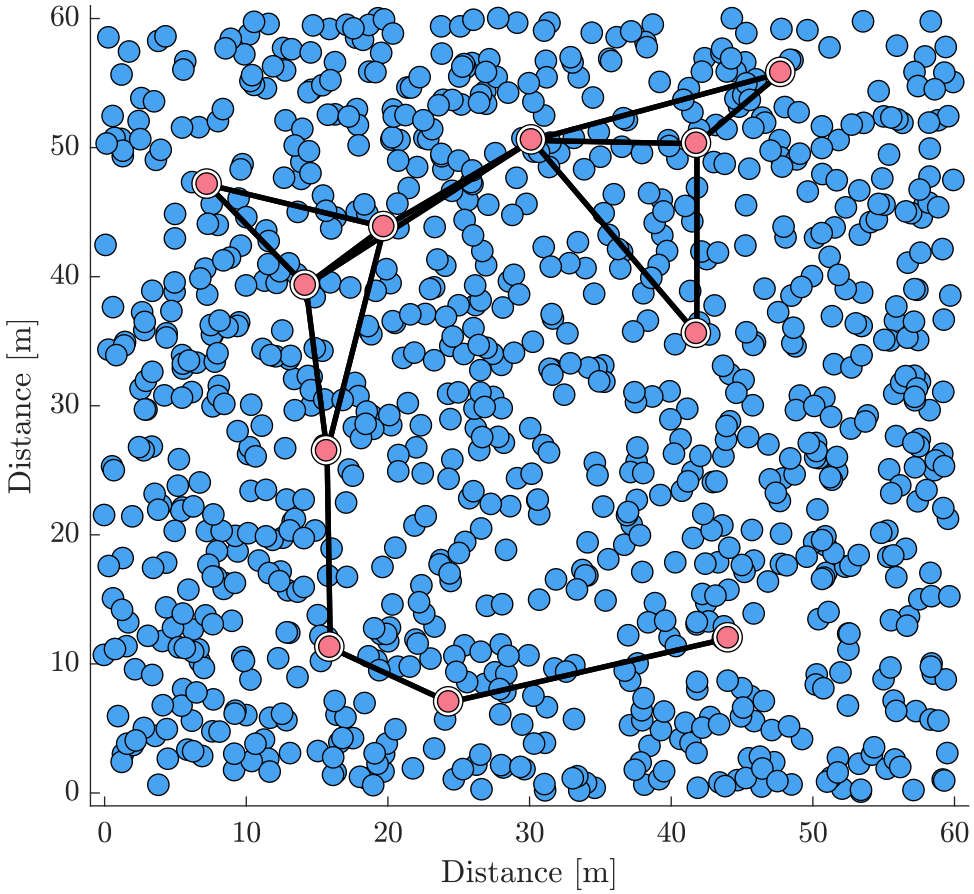


Figure 6.3: The network from Figure 6.1 with the resulting relays distrubution after running the distributed highest connectivity based greedy algorithm marked by red dots. Connections between relays are marked with black lines.

Running the central equivalent algorithm on the same nodes, yields 11 relay nodes as shown on Figure 6.4.

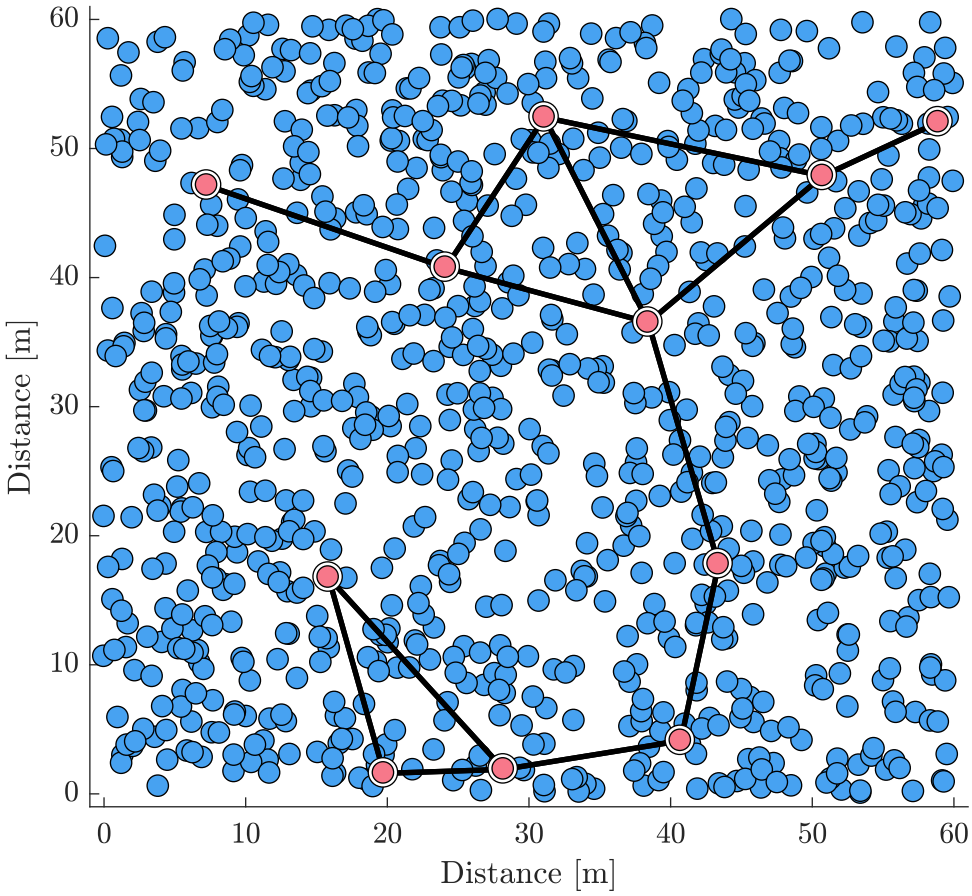


Figure 6.4: The network from Figure 6.1 with the resulting relays distrubution after running the central highest connectivity based greedy algorithm marked by red dots. Connections between relays are marked with black lines.

It is observed that the amount of relay nodes is unchanged, but the placement of the relay nodes changes from the distributed to the central algorithm. The reason for the variation in relay node quantity being this small might be the fact that the average node is in range of approximately 25% of the whole network. This means that the average node can acquire information on nearly the whole network, thus making the distributed algorithm approximately equal to the central algorithms.

6.2 Two hop Pruning algorithm (K2Pruning)

An algorithm which aims to construct a Minimised Connected Dominating Set (MCDS) from a two hop pruning algorithm was developed by Wu and Li [16] and is described in this section. The algorithm does not give satisfying results for dense networks, and is therefore altered by changing when node IDs are checked and by adding an additional step which reduces the Connected Dominating Set (CDS) further.

6.2.1 Description of algorithm

The algorithm aims to construct a MCDS by first making an overly defined set of dominating nodes. Afterwards the algorithm uses two rules to reduce the amount of nodes in the set. For simplicity the algorithm is described in three phases: a construction phase, and two reduction phases.

The algorithm uses two node sets, a marked set D and an unmarked set $G - D$. In the Bluetooth Mesh (BM) typology, marked nodes are equivalent to relay nodes. Before the configuration process starts all nodes are in the unmarked state.

In the **construction phase**, a node u marks itself if a node $v \in N(u)$ exist such that $N(u) \not\subset \big(N(v) \cup v\big)$. This means that if a node has two or more neighbours who are not connected, the node marks itself.

In the **first reduction phase**, a marked node $u \in D$ unmarks itself if another marked node $v \in N(u) \cap D$ exists such that $\big(N(u) \cup u\big) = \big(N(v) \cup v\big)$ and $\mathrm{id}(u) < \mathrm{id}(v)$ or $\big(N(u) \cup u\big) \subset \big(N(v) \cup v\big)$ is fulfilled. If a marked node sees that one of its neighbouring marked nodes covers all its neighbours, the node will unmark itself. If a neighbouring marked node covers the exact same neighbourhood, the node with the lower ID will be unmarked.

In the **second reduction phase**, a marked node $u \in D$ unmarks itself if two other marked nodes $v, w \in \big(N(u) \cap D\big)$ who are also neighbours, exists such that $N(u) = \big(N(v) \cup N(w)\big)$ and $\mathrm{id}(u) = \min(\mathrm{id}(v), \mathrm{id}(u), \mathrm{id}(w))$ or $N(u) \subset \big(N(v) \cup N(w)\big)$ is fulfilled. If a marked node sees that two of its neighbouring marked nodes, who are also neighbours, together spans all it neighbours, the node will unmark itself.

At this point there can still be a lot of redundant relay nodes, a **third reduction phase** is therefore introduced. This phase is similar to the first and second reduction phases, and does the same operation but considers all neighbouring marked nodes at once. This means that a marked node $u \in D$ unmarks itself if any number of marked nodes $\in \big(N(u) \cap D\big)$ who are all neighbours among themselves (a set denoted R), exists such that $N(u) = N(R)$ and $\mathrm{id}(u) = \min(\mathrm{id}(v), \mathrm{id}(R))$ or $N(u) \subset N(R)$ is fulfilled.

Initially the algorithm needs to run at least once on every node in the network, if a node is not marked in the construction phase, the reduction phases will not be executed. Furthermore all nodes should have finished a phase before any node advance to the next phase.

6.2.2 Simulation of Two Hop Pruning algorithm

With the algorithm described, a MATLAB implementation of the algorithm has been made. This implementation can be seen in Code snippet L.1.

Running the algorithm on the network setup seen on Figure 6.1 yields 12 relays as seen on Figure 6.5.

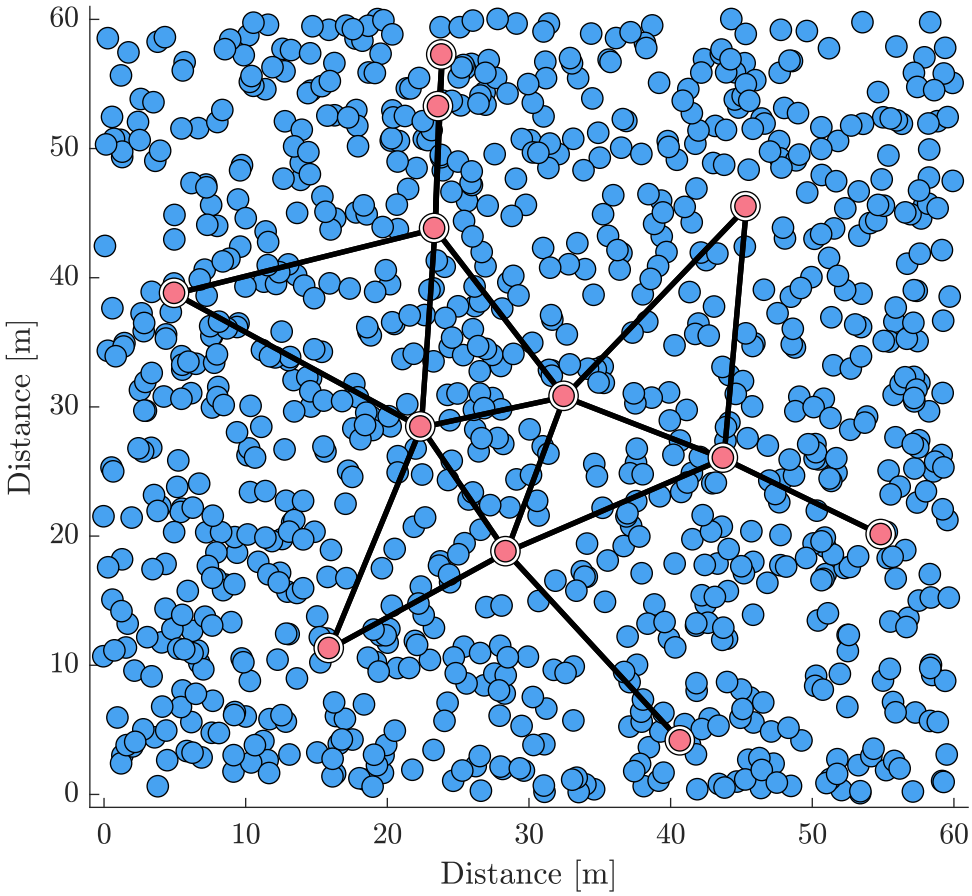


Figure 6.5: The network from Figure 6.1 with the resulting relay distribution from the two hop pruning algorithm marked by red dots. Connections between relays are marked with black lines.

6.3 First come algorithm (FirstCome)

The first come algorithm is based on the saying "first come first served" which means that the node who runs the algorithm first will become a relay node. The first come algorithm will be described in this section. The algorithm is inspired by Wattenhofer's algorithm under the name "Dominator!" [17].

6.3.1 Description of algorithm

The first come algorithm has a construction phase and two connection phases. Similarly to Highest connectivity based greedy algorithm (HighConnect), described in Section 6.1, the first come algorithm uses three sets to describe nodes: a dominating set D, a dominated set $N(D)$, and an undiscovered set $U = G - \big(D + N(D)\big)$.

Before the network configuration starts all nodes are undiscovered. In the Bluetooth Mesh

(BM) network topology dominating nodes are equivalent to relay nodes, the dominated nodes are equivalent to regular nodes and the undiscovered nodes are not part or the mesh network.

In the **construction phase** an undiscovered node u will join the dominating set D, and tell all its neighbours *N(u)* that they are being dominated. This way a node will simply mark itself as a relay if it "comes first".

After the construction phase the first come algorithm will have made a dominating set of relay nodes, this set then needs to be connected.

In the **first connection phase** a dominated node u will join the dominating set D if it is dominated by two different neighbouring dominating nodes v_1 and v_2, but only if no other nodes in the network have announced that they are connecting v_1 and v_2. If node u connects v_1 and v_2 it will announce it to the network.

In the **second connection phase** two dominated nodes u_1 and u_2 will join the dominating set D if two criteria are satisfied. The first being that each of the nodes u_1 and u_2 are dominated by different nodes v_1 and v_2, such that $v_1 \in (N(u_1) \cap D) \vee v_1 \notin N(u_2)$ and $v_2 \in (N(u_2) \cap D) \vee v_2 \notin N(u_1)$. The second criteria being that, node v_1 and v_2 have not already been announced to be connected. If node u_1 and u_2 connects v_1 and v_2 it will announce it to the network.

This means that nodes must share information on what relays they are dominated by with their neighbourhood before the second connection phase can run.

In the two connection phases, redundant relay nodes can be created if one or two nodes u_1 and u_2 connects a pair of dominating nodes v_1 and v_2, but all the dominating nodes neighbours $N(v_1) \wedge N(v_2)$ does not receive the connection announcement. In this case multiple nodes can connect the same dominating nodes, creating redundant relay nodes.

In the connection phases a rule is added to avoid that multiple nodes connects the same dominating nodes, due to them not being in range of each others. The rule is as follows.

- If a dominating node receives an announcement that is has been connected by another node, it will relay the announcement.

This way all neighbours of a pair of dominating nodes v_1 and v_2 receives connection announcements.

Furthermore, it is chosen to implement the algorithm so that only the original dominating set will be considered when connecting dominating nodes in the two connection phases. Because if a node u is marked as a relay by the first connection phase, u will not need to be connected any further. No node should change state in order to connect u to the other relay nodes.

These two modifications of the proposed algorithm will result in fewer relay nodes, yet still guaranteeing full node coverage. This is done because it was seen that otherwise the algorithm would make almost all nodes relay.

Initially the algorithm needs to run at least once on every node in the network. Furthermore

all nodes should have finished a phase before any node advance to the next phase.

Simulation of first come algorithm

With the algorithm described a MATLAB implementation of the algorithm have been made. The algorithm is seen in Code snippet M.1.

Running the algorithm on the network setup seen on Figure 6.1 yields 29 relays as seen on Figure 6.6.

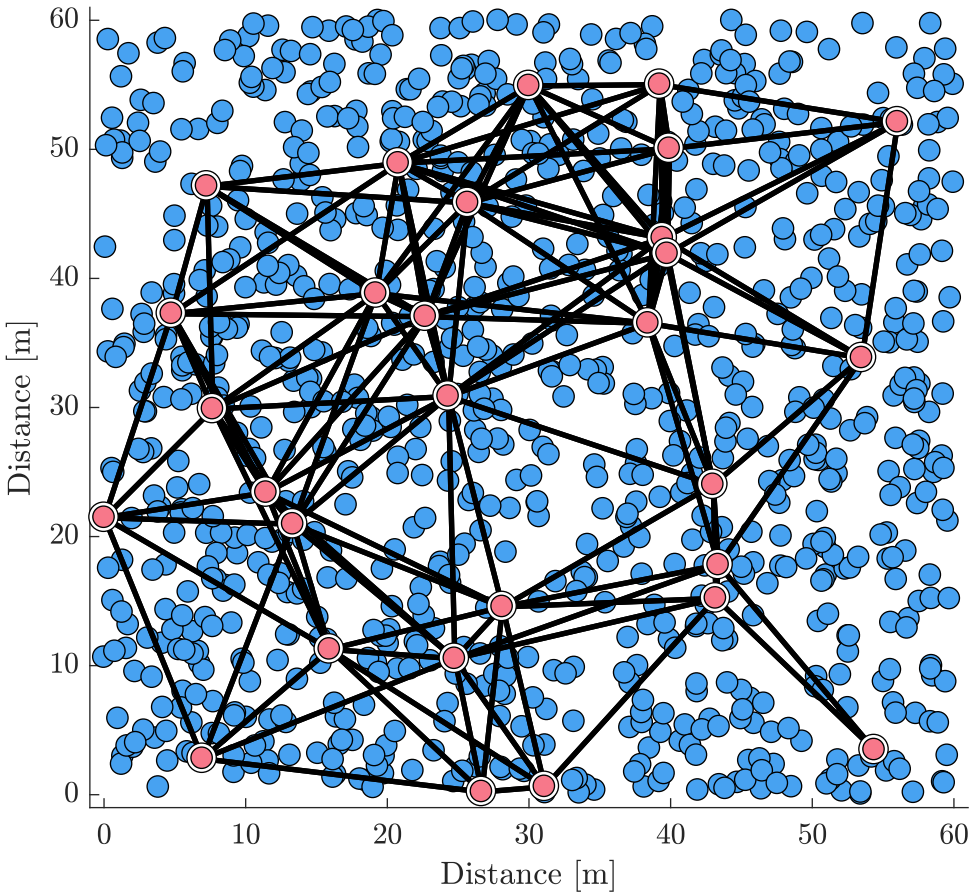


Figure 6.6: The network from Figure 6.1 with the resulting relays distrubution from the first come algorithm marked by red dots. Connections between relays are marked with black lines.

6.4 All nodes are relays

The last distribution used in the comparison is all nodes are relays.

With all nodes as relays the network will be guaranteed to have full coverage and it will have a high level of redundancy, however at the cost of the maximum amount of packets

and retransmissions.

6.5 Evaluation of algorithms

With five relay distribution algorithms described, multiple simulations are run. These simulations can then be used to get a picture of how the algorithms perform in regards to different performance metrics. First the memory usage on the nodes and the required quantity of transmissions for setup is approximated. Then the simulator is run and the Packet Delivery Ratio (PDR) of the networks configured by the algorithms is measured. Finally, a worst case scenario is set up for each algorithm and the PDR is simulated. The parameters used for simulations throughout this chapter can be found in Appendix E.

6.5.1 Needed transmissions for setup

The algorithms use different amounts of information from their neighbours. In this section, an estimate is given for the number of messages needed to distribute relays in the network. The estimate is found by assuming no packet loss between neighbouring nodes. The values used are given:

- n Amount of nodes in the network.
- Δ Average neighbour count.
- Δr Amount of relays in the resulting network.

The calculations in this section serves as simple approximations. Data could be distributed in a smarter way than just flooding uncompressed data throughout the network. The packet quantity might also be reduced through implementation specific methods. This section will not consider such methods but instead use flooding of uncompressed data.

The **HighConnect** algorithm finds out how many nodes their neighbours can see. If a node can not see a relay then it announces that it is a relay, or that some other node should be relay. This accounts for Δr messages.

To connect the dominating set, clusterID's are synchronised. The clusterID's are propagated throughout the whole network by using relays to distribute them.

When a node joins the dominating set, the clusterID's must be propagated. Because clusterID's only needs to be propagated through the whole Connected Dominating Set (CDS) every time a higher clusterID is found, the worst case would be if every time a node joins the CDS, it does so with a higher clusterID than the rest of the CDS. This would cause the clusterID of the newest relay to propagate throughout the whole CDS every time a new node joins the CDS. This will cause the upper bound to be $\sum_{i=1}^{\Delta r} i$.

Nodes must transmit an update of their status every time their neighbourhood changes. It is assumed that every new relay changes every nodes neighbourhood.

- n For telling, 'I am here'.

- $n \cdot \Delta r$ For telling neighbours, 'My status is..'.
- Δr For telling, 'You should be relay' and 'I am relay'.
- $\sum_{i=1}^{\Delta r} i$ For propagating CID.

The **CentralHighConnect** algorithm needs a list of neighbours for all nodes. Since messages needs to be sent even if there is only one neighbour, the amount of messages is rounded up to the nearest integer. All messages and information about nodes needs to be gathered at one central node. Since no routing is implemented and no relay nodes are chosen, all packets are relayed by all nodes.

When the central node decides which nodes should be relay, it sends messages out to the chosen nodes.

Packets can contain 16 B of payload, seen from the network layer[1], and an ID is 2 B. This means that a packet can contain 8 IDs.

- n For telling 'I am here'
- $n \cdot \left\lceil \frac{\Delta}{8} \right\rceil \cdot n$ For telling, 'I can see these nodes'.
- $\Delta r \cdot n$ For telling, 'You should be relay'.

For the **K2Pruning** algorithm, each node needs a list of neighbours from all neighbours. If the node can see that it should not be a relay, through some rules, it announces this. The quantity of nodes leaving the dominating set is denoted x, where $x < n - \Delta r$.

- n For telling, 'I am here'.
- $n \cdot \left\lceil \frac{\Delta}{8} \right\rceil$ For telling, 'I can see these nodes'.
- $\Delta r + x$ For telling, 'I am a relay'.
- x For telling, 'I am not a relay any more'.

The **FirstCome** algorithm needs to tell neighbours that they are being dominated. Then, for connecting the dominating set, all nodes say which relays they can see. If nodes connects relays, they tell their neighbourhood. The quantity of nodes joining the dominating set, to connect the network is denoted $x < n$.

- $\Delta r - x$ For telling, 'I am a dominator'.
- n For telling, 'I can see these relays'.
- x For telling, 'I connect relays'.
- $> \Delta r$ For telling, 'I have been connected to relay ID..'.

Calculating with different network setups, different algorithms will perform better. In Table 6.1 the amount of configuration messages are given for different network topologies.

[1] see Section 2.5

Table 6.1: Configuration time shown as number of packages that needs to be sent. For each set of parameters, the two algorithms with the lower transmission quantities are marked with grey backgrounds.

Parameters				Results [packets]			
n	Δr	Δ	x	HighConnect	CentralHighConnect	K2Pruning	FirstCome
1000	200	10	100	221300	2201000	163800	2200
1000	50	100	25	52325	13051000	15950	2050
1000	1	1000	0	2002	125002000	127998	2001

From Table 6.1 it is seen that **HighConnect** and **FirstCome** requires less packets when the network is dense. When the network is sparsely connected, **K2Pruning** and **FirstCome** requires fewer packets than the rest.

6.5.2 Storage usage

The Bluetooth Mesh (BM) will have a limited amount of storage to hold the information needed in order to distribute relays. In this section, an estimate of how much space needed for each algorithm is found.Nodes could use compression algorithms to reduce the amount of storage necessary. Since a node generally does not have a large quantity of processing power, this is not looked into.

The **HighConnect** algorithm needs nodes to store information about their neighbourhood. The information needed on a neighbour is listed in Table 6.2.

Table 6.2: Amount of bits used to store different information

ID	Unconnected Neighbour Count	ClusterID (only for relays)	Total
16 b	16 b	16 b	6 B

Each node has to maintain a list for each neighbour, therefore the storage needed for the individual node is approximately given by Equation (6.1)

$$\text{HighConnect}_{\text{storage}} = (\Delta - r) \cdot 4\,\text{B} + r \cdot 6\,\text{B} \tag{6.1}$$

Where:

Δ	is the average amount of neighbours	[1]
r	is the quantity of relays the average node can see	[1]

The **CentralHighConnect** algorithm uses information that is gathered on all nodes. This means that the central node needs all information, and the single node needs a minimum amount of information. If the individual node sends information to the central node right after getting the information, then the space needed on the individual node will be minimal.

The central node needs a map of the complete topology. So for each node n, there is Δ neighbours. Every ID is 2 bytes, so the storage needed is approximately given by

Equation (6.2)

$$\text{CentralHighConnect}_{\text{storage}} = n \cdot \Delta \cdot 2\,\text{B} \tag{6.2}$$

Where:

n	is the amount of nodes	[1]

The **K2Pruning** algorithm uses neighbours of neighbour information. This means that the individual node needs to keep a vector of neighbours for each neighbour. This is approximately given by Equation (6.3)

$$\text{K2Pruning}_{\text{storage}} = \left(\Delta^2 + \Delta\right) \cdot 2\,\text{B} \tag{6.3}$$

The **FirstCome** algorithm uses knowledge about neighbouring and neighbour's relays, this gives approximately the storage of Equation (6.4).

$$\text{FirstCome}_{\text{storage}} = (\Delta \cdot r + r) \cdot 2\,\text{B} \tag{6.4}$$

The storage space used for the different algorithms are summarised in Table 6.3 where different usecases are given.

Table 6.3: Storage space used by the different algorithms using different parameters.

Parameters			Results [B]			
n	Δ	r	HighConnect	CentralHighConnect	K2Pruning	FirstCome
1000	10	3	46	20000	220	66
1000	100	3	406	200000	20200	606
1000	1000	3	4006	2000000	2002000	6006

It can be seen from Table 6.3 that the **HighConnect** algorithm uses the least amount of storage across the different usecases. **FirstCome** is however a close second.

6.5.3 Simulation of PDR results

To find the performance of the different algorithms, simulations done with the node setup shown in Figure 6.1. As mentioned previously, Appendix E covers the parameters, data processing, and code used in the following sections.

For evaluating the performance of the different algorithms, each algorithm is simulated for two different node densities: dense and sparse. For both node densities, the network is simulated with three different node layouts. The node layouts, densities, and traffic is kept the same for all algorithms, to make a fair comparison. The PDR is measured as a function of the traffic intensity, and averaged for the three simulations with identical node densities. The results shown in Figure 6.7, in which it can be seen that HighConnect, K2Pruning, and CentralHighConnect works well. In the sparse simulation seen in Figure 6.8, K2Pruning works a little better than HighConnect and CentralHighConnect.

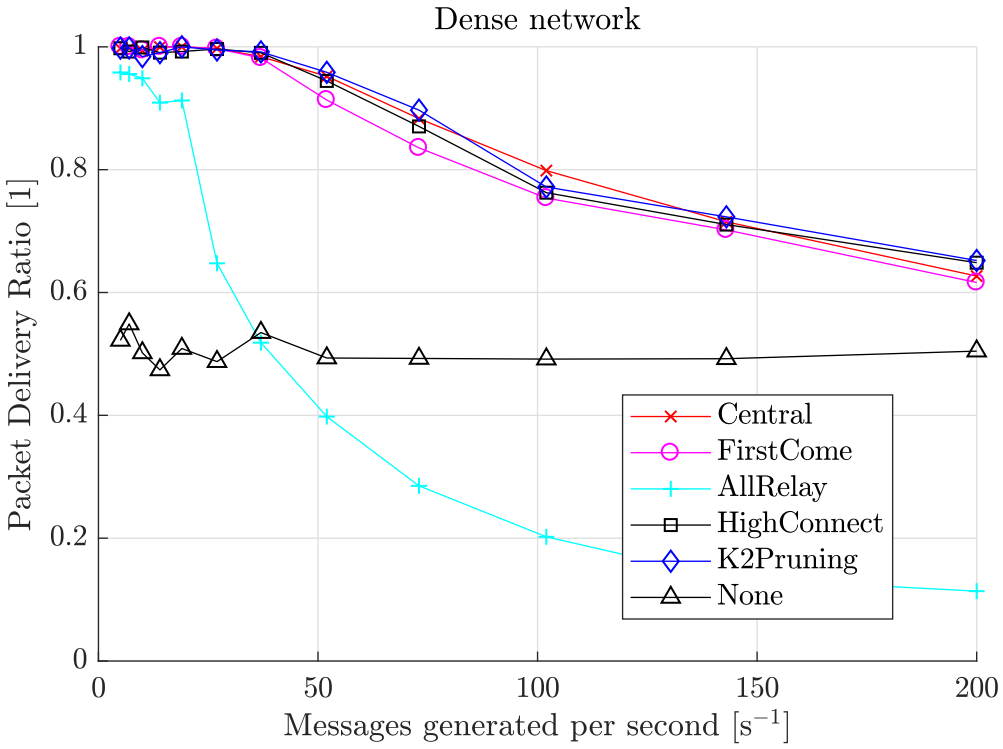


Figure 6.7: Average PDR at a different amount of messages per second for the dense network. The network has 1000 nodes placed uniformly by MATLAB's `rand` function in a 60 m by 60 m area.

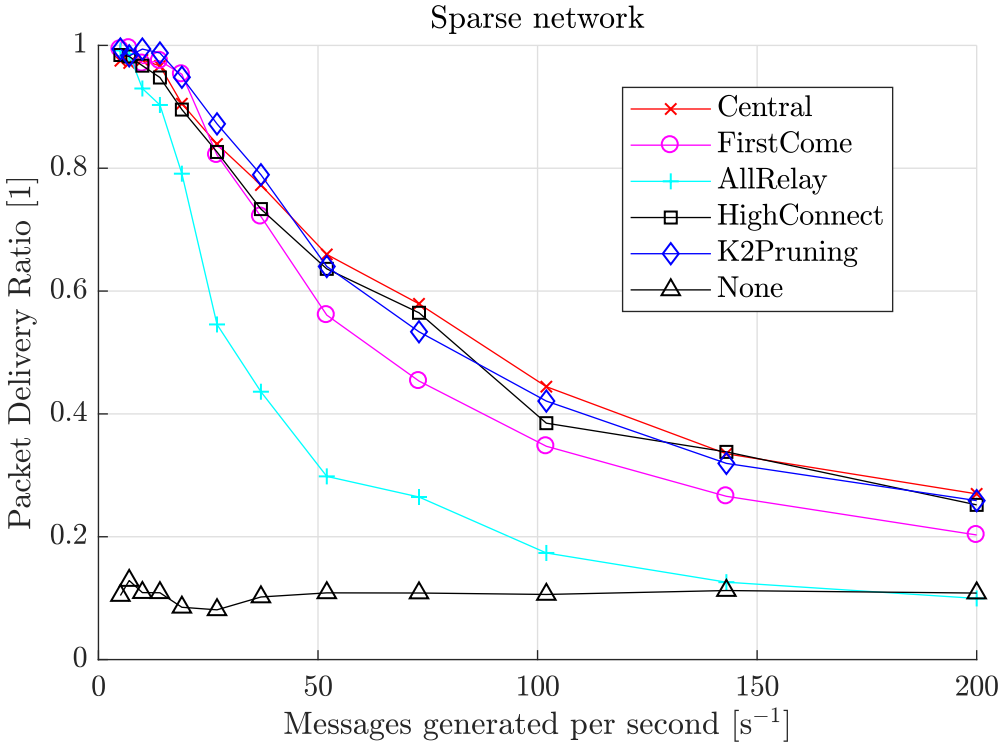


Figure 6.8: Average PDR at a different amount of messages per second for the sparse network. The network has 1000 nodes placed uniformly by MATLAB's `rand` function in a 150 m by 150 m area.

It is seen from Figures 6.7 and 6.8, that there are small differences between the algorithms, but they all outperform AllRelay and no relays. As the traffic intensity increases, the FirstCome algorithm starts to fall behind the other algorithms.

6.5.4 Worst case PDR simulation

PDR does not give the complete picture of performance regarding the resultant network. Since the network first of all should seek to ensure that all nodes are connected, any node should be able to communicate to any other node with an acceptable PDR. To find out which algorithm provides the best PDR even at worst case scenarios, it is decided to track messages between nodes with the longest euclidean distance between them. The 10 node pairs with the highest euclidean distance between them are visualised on Figure 6.9.

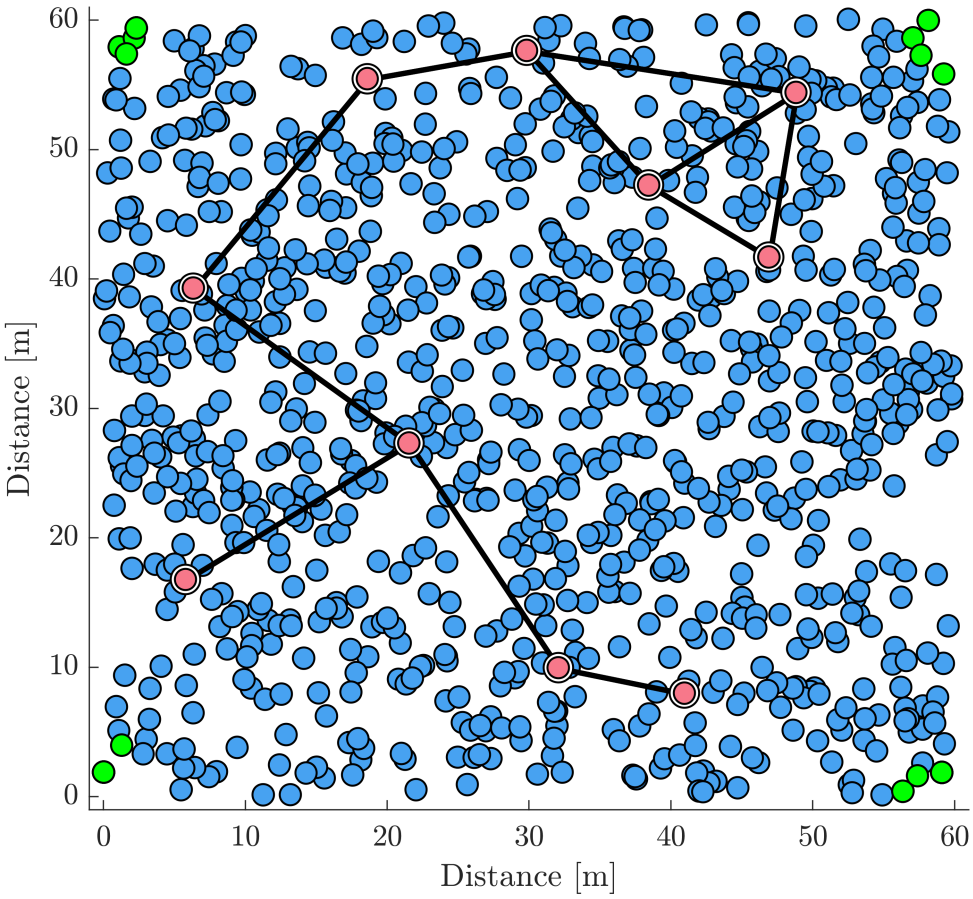
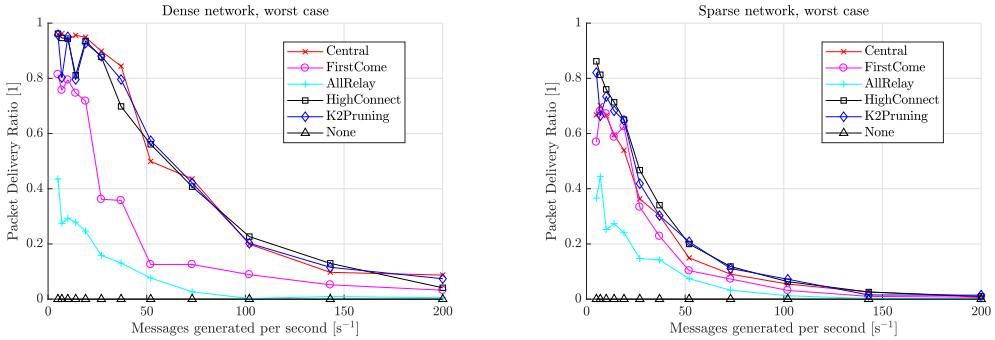


Figure 6.9: Illustration of a dense network after running the HighConnect algorithm. The node layout is created using a different random number generator seed than previously in this chapter. The green nodes are the nodes with the longest euclidean distance between them.

In order to test the PDR, 10 messages are sent per second between the node pairs with the longest euclidean distance between them. After the simulation is over, the messages sent and received is counted. Other than focusing on the "worst case" nodes, these simulations follows the same methodology as used in the previous simulation in Section 6.5.3. This means that simulations are run at multiple different traffic intensities (for the rest of the network) and with different network areas. The rest of the network will therefore be communicating, likely causing collisions and interference.

The results are given in Figure 6.10.



(a) Dense network ***(b)*** Sparse network

Figure 6.10: Worst case PDR with different network areas, algorithms and traffic intensities.

From Figure 6.10 it can be seen that the CentralHighConnect, HighConnect and K2Pruning algorithms are almost identical in worst case PDR. When there are no relays, the PDR is equal zero, since the networks are unconnected. AllRelay and FirstCome is found to perform much worse than the alternatives in the dense networks, while only AllRelay stands out as being worse in the sparse networks.

Because of the algorithms K2Pruning and HighConnect have shown to perform the best in the different cases, while being fully distributed algorithms, they will be analysed further. Additional worst case simulations are run with a wider range of different network scenarios to get a better look at how the algorithms perform overall. Figure 6.11 displays the results. Each node density is simulated with 12 different layouts, totalling up to 48 different scenarios for each algorithm. The 12 different layouts are the same for both algorithms and the traffic is exactly the same for all simulations, to make for a fair comparison. All networks contains 1000 nodes.

From Figure 6.11 it can be seen that K2Pruning performs better overall when the network is sparsely connected. This might be due to the fact that the K2Pruning algorithm creates a more interconnected network, while HighConnect is more likely to create a few paths connecting the whole network. As the network area becomes large, the length of few paths connecting the network can become long. Figure 6.12 shows the relay distributions after running the algorithms on a 330 m by 330 m network with 1000 nodes.

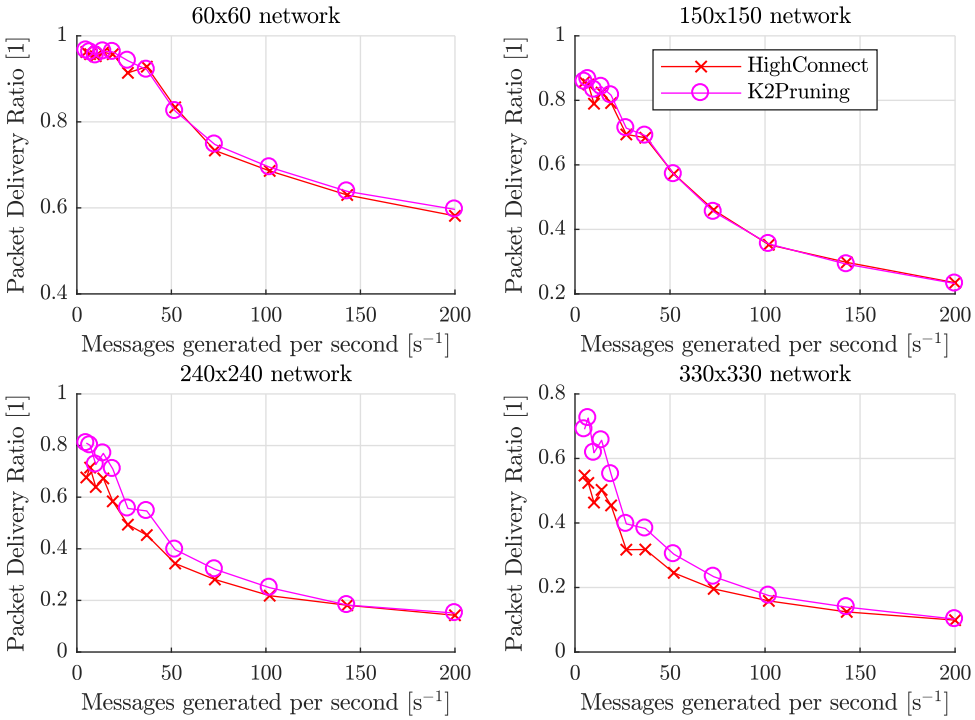
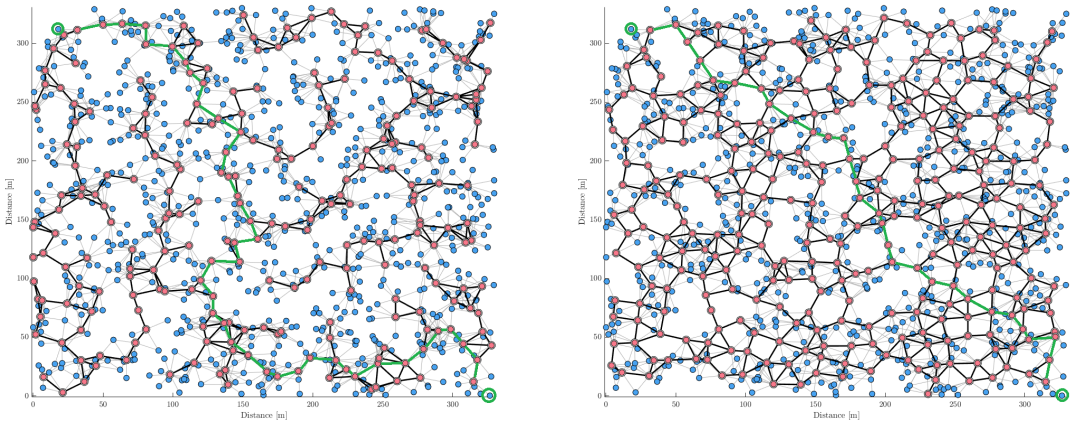


Figure 6.11: Graphs showing the simulated "worst case" PDR for the K2Pruning and HighConnect algorithms across different network areas.



(a) HighConnect ***(b)*** K2Pruning

Figure 6.12: Node distributions for a 330 m by 330 m network. Fullsize figures can be found in Appendix D. The path consists of 44 jumps for HighConnect, while the path is 33 jumps in K2Pruning.

Figure 6.12 shows how the K2Pruning algorithm creates a very interconnected mesh network, while HighConnect has a few long paths connecting the whole network. A path between two nodes with a long euclidean distance is marked on both networks. The length of the path is 44 jumps in the HighConnect network, while the path is 33 jumps in the K2Pruning network. It can also be seen from Figure 6.12 that there are more paths to

the destination in the K2Pruning algorithm. These observations might be the reason K2Pruning outperforms HighConnect in the sparse networks.

For dense networks there is no observable difference between the worst case PDRs of the algorithms. Looking at this and the approximated storage usage in Section 6.5.2, the HighConnect algorithm appears as a good compromise between network performance and node complexity.

6.6 Conclusion of algorithm analysis

Throughout the algorithm analysis chapter, different algorithms have been described and simulated with different test scenarios for comparing them.

In the beginning of this chapter the different algorithms are described and examples of configured networks is shown. Each of the algorithms found a Connected Dominating Set (CDS).

After analysing the algorithms, different networks configured by the algorithms are simulated in the network simulator from Chapter 5. When distributing relays, all nodes are assumed to have updated and complete information about their neighbourhood or the whole network, depending on the used algorithm. All algorithms perform better than the baseline that is all relays.

Average Packet Delivery Ratios (PDRs) are simulated for all algorithms to compare the average performance of the networks after running the different algorithms. Worst case PDRs is also simulated to find out which algorithms performs the best under these conditions. The worst case PDRs also displays if the network is connected, since an unconnected network will have the lowest PDR of 0.

The worst case PDR tests showed that K2Pruning, CentralHighConnect and HighConnect performs the best. Since there is no significant difference between the CentralHighConnect and HighConnect algorithms, the CentralHighConnect is discarded since it requires much more information and packets to get similar results. Following this, K2Pruning and HighConnect worst case scenarios are compared. From the worst case test it can be seen that K2Pruning is the algorithm that provides the best results in sparse networks, as seen in Figure 6.11. However for dense networks the results are very similar.

K2Pruning might perform better in sparse networks due to the fact that is has more and shorter paths to a destination. K2Pruning however requires large quantities of data to be stored on the nodes, and offers very similar performance to the HighConnect algorithm in dense networks.

HighConnect is therefore chosen for implementation in Chapter 7 since it appears as a good compromise between network performance and node complexity.

Algorithm implementation 7

As described in Section 6.6, a distributed greedy algorithm favouring high connectivity, showed satisfying results. In this section the algorithm described in Section 6.1 will be implemented on nodes in the network simulator.

In Chapter 8 the algorithm running distributed on the nodes will be tested with respect to the performance metrics described in Chapter 4.

When implementing the algorithm, a range of problems must be solved. Due to limited project time and long simulation times, the implementation is only designed for a 60 m by 60 m area with 200 nodes. Lookup tables and curve-fits could be made for a range of different node densities and areas, so the algorithm would work in other networks. It is assumed that the nodes knows the network area and node quantity when the setup begins. Nodes keep track of time perfectly in the simulator, which is not necessarily true. However this is assumed to amount to insignificant errors with the time scales used in this simulation. This project does only consider static networks, therefore network maintenance not be discussed or implemented.

7.1 Implementation specific problems

When implementing the algorithm in a distributed fashion, the assumptions from Chapter 6 cannot be used. This means that nodes have to transmit packets containing information about themselves and their neighbourhood. Nodes can therefore have incomplete information about their neighbourhoods due to packet loss or unfortunate random events. Multiple nodes can make conflicting decisions simultaneously if no counter measures are implemented, leading to redundant relay nodes.

The algorithm works based on neighbourhood information. This however poses a problem when implementing the algorithm since the range of a node is dependent on the noise within the network. The noise and interference within the network is dependent on the quantity and distances of transmitting nodes. No outside noise is considered, although it would have a very real impact on the performance of the network (see Chapter 5). The noise, and by extension neighbour range, varies over time. This makes defining the range of a neighbourhood difficult. Figure 7.1 shows the packet arrival probability depending on noise and distance between sender and receiver.

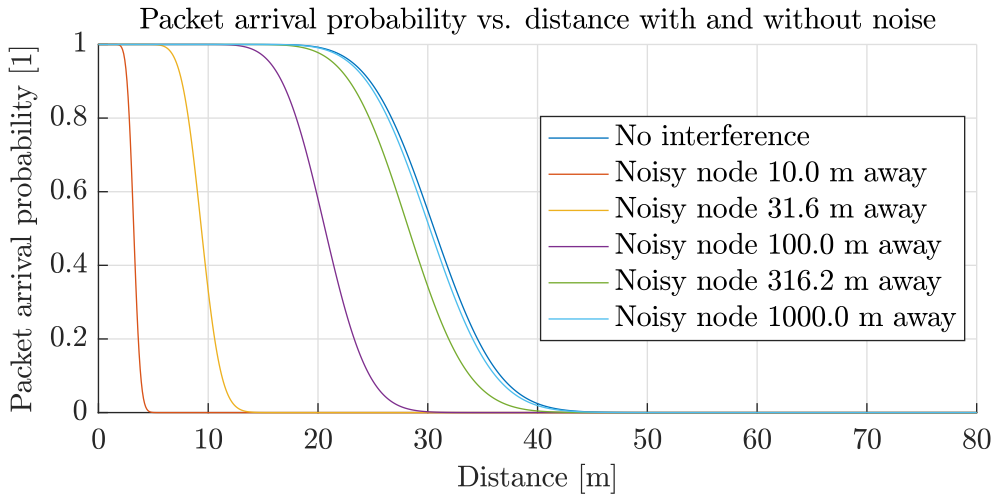


Figure 7.1: Graph showing the packet arrival probability at different noise levels with distance between the sender and receiver on the x-axis.

Figure 7.1 shows that the packet arrival probability distance varies greatly with noise. If the network is set up while the noise is high, the nodes will have short range and the quantity of relay nodes will increase. When the noise is low, the nodes will have a longer range and the quantity of relay nodes will decrease. This can result in suboptimal relay placement if the network is set up with noise that is not representative of the noise in ordinary operation. If the noise during set up is very low compared to noise in ordinary operation, the nodes can in worst case be out of range of all neighbouring relay nodes. Since nothing can be said about what is the actual amount of traffic on the network, neighbour discovery could be done dynamically during normal operation. This is further discussed in Chapter 9.

When implementing the algorithm the following problems must therefore be considered:

1. Discovery of neighbourhood.
2. Avoiding simultaneous conflicting decisions.
3. Synchronising setup phases.
4. Lowering risk of making decisions based on incomplete or outdated information.

7.2 Solution proposal

To solve these implementation specific problems a two stage solution is proposed: a neighbour discovery stage and a relay distribution stage. In the discovery stage nodes will determine how many neighbours they have and their IDs. In the second stage nodes start sharing information about themselves and make decisions to configure the network. The relay distribution stage will follow the phases of the distributed highest connectivity based greedy algorithm described in Section 6.1.

Nodes must not make conflicting decision at the same time. This can be avoided if two nodes in the same neighbourhood do not make network changing decisions at the same time. As described in Section 6.1 the network configuration has multiple phases. It is

important that nodes wait on each other. Nodes should not progress faster than the rest of the network. A node can only start the next phase when all other nodes in its neighbourhood are ready. This means that nodes will not start the connection phase before the construction phase is finished. To avoid nodes advancing too fast a rule is implemented, a node does not advance further than one step above the lowest step of all its neighbours and neighbours of neighbours.

In the discovery stage the nodes injection rate is chosen based on what results in neighbourhoods close to the ideal case from Section 6.1. In the relay distribution stage the nodes injection rate is optimised to get the minimum time between packet receipt among neighbours, so the network can be set up quickly.

Before the discovery and relay distribution stages are described, the information that is shared among nodes and how it is shared is presented.

7.3 Sharing information among nodes

Information can be shared on a ask and receive basis or by flooding the network. Sharing information on a ask and receive basis can result in a higher packet count due to the need for control packets, and the node complexity would increase since it has to keep track of which of its neighbours has or has not replied with updated information. If the node does not have to ask for the information, less control packets will be send. If the node does not ask for the information, it is however difficult to detect if packets are lost. Waiting for an information reply can potentially take a long time, thus extending the setup time.

This project chooses to share information by flooding the network continuously. Nodes will periodically broadcast packets in order to inform potential neighbours about their presence in both phases. Using this method, the probability of nodes making decisions based on old data should be higher due to packet loss, potentially resulting in redundant relay nodes. It however is assessed that the reduced setup time and complexity outweighs the increased probability of old data.

In order to make nodes able to share information about themselves with their neighbours an Network Setup Packet (NSP) is constructed. During the network configuration, nodes only do three main tasks. They send, receive, and make decisions based on the received NSPs. Next, the content of the NSP is introduced and described. Afterwards the purpose of the chosen content is discussed.

As described in Section 6.1, the highest connectivity based greedy algorithm requires nodes to inform their neighbours about:

- Their role (undiscovered, dominated or relay).
- How many unconnected neighbours they have.
- What their cluster ID is.
- If a neighbour should change its role to relay.

Because the chosen algorithm requires this information, it is included in the NSP. The NSP has a total size of 10 B. The content of the NSP is listed in Table 7.1.

Table 7.1: Content of NSP.

Name	Size [b]	Description
`UNC`: Undiscovered neighbours	16	Amount of Undiscovered neighbours.
`CID`: Cluster ID	16	Highest Cluster ID of a neighbouring relay.
`LNS`: Lowest neighbour step	3	The lowest step of a neighbour.
`Role`	2	Role of the node.
`Step`	3	Step of the node.
`BR`: Be relay	16	The address of a neighbour node, who the node have marked as to become a relay.
`PPS`: Packets per second	8	The amount of packets the node sends per second.
`PS`: Packets sent	16	The amount packets sent in current stage.

The NSP is sent with a header which includes sequence number[1], source address, destination address, Time To Live (TTL), and a total message length indicator. For NSPs the source address[2] is set as -1 and the TTL is set as 0. Using these markers, the receiving nodes can easily identify if a packet is an NSP and they will not travel outside the neighbourhood.

With the headers included, the total packet size[3] is $10\,\mathrm{B} + 31\,\mathrm{B} = 41\,\mathrm{B}$.

`BR` is used if a node decides that one of its neighbours should be a relay. If node u decides that node v should be a relay node u will put the address of node v in the `BR` field of its NSP.

`LNS` allows a node to see how far its neighbours of neighbours are in the setup phases. This allows the nodes to avoid making decisions simultaneously with nodes in their two hop neighbourhood. Three hop neighbourhood could be added to give more knowledge about the network. Two hop neighbourhood information is chosen as a compromise between complexity, configuration duration, and probability of nodes making decisions conflicting decisions. If nodes have to wait for the three hop neighbourhood before making decisions, the configuration duration will be increased since fewer simultaneous decisions happens.

The `PPS` field indicates the current rate of NSP transmissions, in packets per second, of the node. *This value was meant to be used for a feature that was not implemented. This value was meant to enable the individual node to calculate the injection rate within its neighbourhood. The nodes could then adjust the injection rate to an optimised value where the time between packet receipt among neighbours is minimised. This value was however by mistake not removed from the NSP, increasing its size by 1 B.*

The NSP contains a counter called `PS`. This field contains the quantity of packets send in the current stage. This allows nodes to calculate a packet delivery ratio based on this number and the total quantity of received packets in the current stage.

[1] In the network simulator the sequence number is abstracted into a packet ID

[2] The value -1 is chosen since it can not collide with other IDs in the simulator.

[3] In a Bluetooth mesh packet there is 31 B header per packet and the total length of a transmission can be 47 B, see Section 2.3 and Section 2.5.

In the discovery stage NSPs are used to tell neighbours about their presence. In the relay distribution stage NSPs used to decide upon a set of delay nodes. When an NSP is received the content of the packet, together with other information about the packet sender, is saved in a matrix in the receiving node. This matrix is called the `NeighbourMatrix`. If a node receives an NSP from a node it has already received an NSP from before, the receiving node updates the row of the `NeighbourMatrix` assigned to the packet's source.

The full content of the `NeighbourMatrix` is seen in Table 7.2.

Table 7.2: Neighbour matrix

Entry	**Name**	**Size [b]**	**Description**
1	`ID`	16	Address of packet sender (source address of packet)
2	`UNC`	16	Amount of Undiscovered neighbours.
3	`CID`	16	Highest Cluster ID of a neighbouring relay.
4	`LNS`	3	The lowest step of a neighbour.
5	`Role`	2	Role of the node.
6	`Step`	3	Step of the node.
7	`BR`	16	The address of a neighbour node, who the node have marked as to become a relay.
8	`PPS`	0	The amount of packets the node sends per second. (deprecated)
9	`PS`	16	The amount packets sent in current stage.
10	`RSSI`	0	Received Signal Strength Indicator (deprecated)[4]
11	`MC`	16	Message Counter. Counts the quantity of packets received from a node in the current stage

The `NeighbourMatrix` will at most be of size equal to the amount of neighbours the node has multiplied by 14 B.

7.4 Stage 1: Discovery

In the discovery phase nodes need to determine how many neighbours they have. Three methods of discovery are considered:

1. Nodes are neighbours if a setup packet has been exchanged between them.
2. Nodes are neighbours if the Received Signal Strength Indicator (RSSI) is within a threshold during the duration of a successfully received packet.
3. Nodes are neighbours if the Packet Delivery Ratio (PDR) between two nodes is above a threshold.

When testing for whether a node is a neighbour, there can be four different outcomes as listed in Table 7.3.

[5]To get a more accurate estimate of whether a node is a neighbour or not, one could use the RSSI in conjunction with the Packet Delivery Ratio (PDR) used in the Discovery phase

Table 7.3: Table listing the different outcomes when determining whether a node is a neighbour or not. Inspired by [18, Lecture slides page 9]

Node is neighbour	Accepted	Outcome
True	Yes	Ok
True	No	Type 1 error. Node sees less neighbours, results in redundant relays.
False	Yes	Type 2 error. Node sees too many neighbours, results in weak connections between relays and nodes.
False	No	Ok

Since type 2 errors can cause nodes to accept relays where communication has a very high Block Error Probability (BLEP), causing an unconnected network, they are deemed of greater cost than type 1 errors. For this reason, type 2 errors should be avoided while ideally still keeping type 1 errors low.

Errors are determined by comparing the distance between the nodes with the 0.95 PDR range assuming no interference. This distance is calculated using the path loss and BLEP for Additive White Gaussian Noise (AWGN) channels from Section 5.2, as $d_{\text{Threshold}} \approx 23.18\,\text{m}$.

First discovery approach

Using the first discovery approach, there will be close to zero type 1 errors, but a significant quantity of type 2 errors. This is due to there still being a relatively high probability of receiving packets from out-of-range nodes. Figure 7.2 shows the Probability Mass Function (PMF) of the distance between the source and destination nodes, given that a packet has arrived without errors. The receiving node is in the middle of a circular shaped network with radius $r = 50\,\text{m}$. The rest of the nodes are placed randomly within the network using a random uniform distribution.

As is shown on Figure 7.2, the node can still receive packets from out-of-range nodes, and will most likely do so often. If the first approach is used, the node will make a type 2 error every time the node receives a packet from a new out-of-reach node.

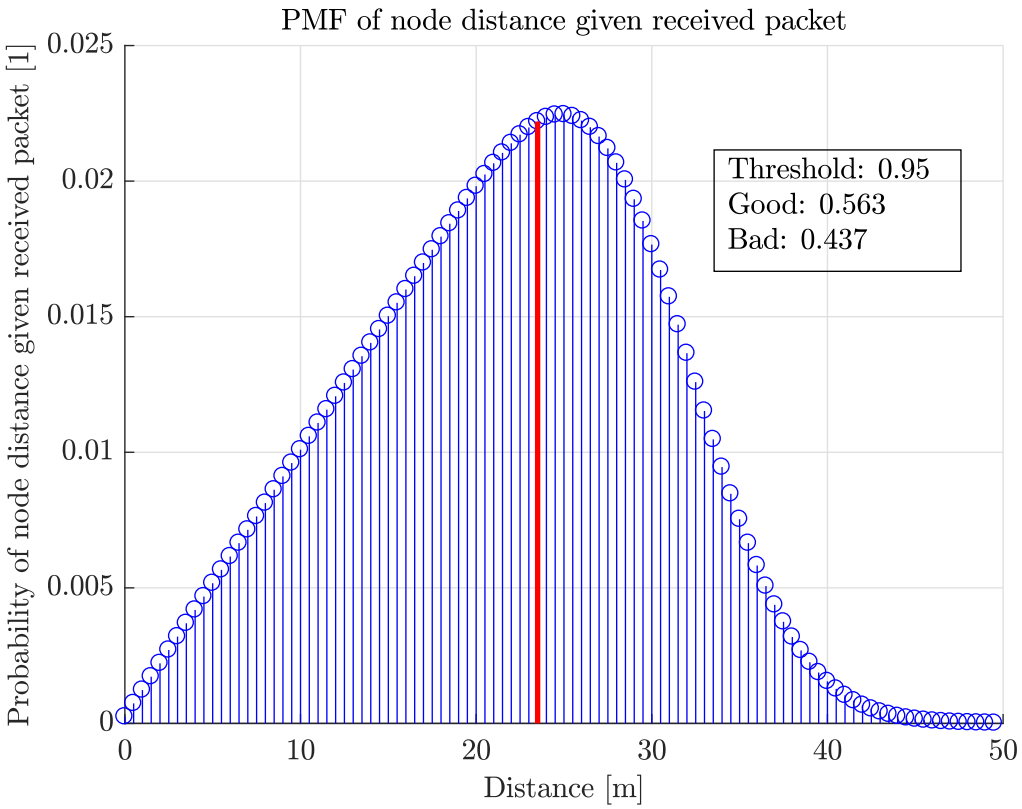


Figure 7.2: Graph of the theoretical PMF of the distance between two nodes given that a setup packet has just been received, assuming no interference. The value at a point is the probability of a node being within the point and the previous.

Second discovery approach

The second approach for discovery is using the RSSI value read from packet receipt. The Bluetooth Core 5 standard specifies how precise the RSSI reading for a bluetooth LE device is. It returns a value of absolute receiver signal strength in dBm with ±6 dBm accuracy [1, p. 1066]. The range that can be returned is between -127 and 20 in dBm.

Since the standard does not specify how the error of the RSSI is distributed, it could be heavily skewed. If the RSSI is skewed, then that means that taking the average over packets will not give the true result. Furthermore, noise on the channel will cause the RSSI to become larger than the real value, causing nodes to make type 2 errors.

The RSSI value could be used to filter out edge cases where a message is received even though it is very far away. To get a precise reading of PDR, the RSSI would either need to be calibrated or used it in conjunction with another method.

Third discovery approach

The third approach is to approximate the PDRs and compare it to a threshold. Since the PDR is ultimately what defines which nodes are neighbours and which nodes are not, it is obvious to use this as a measure. Measuring the PDR precisely does require a lot of

samples, which takes time to gather. If the channel is noisy the PDR will fall, causing a node to make more type 1 errors and less type 2 errors. Since type 2 errors are deemed of higher cost, this is a desired property.

The third discovery approach of measuring PDR is chosen since noise does not make the nodes make more type 2 errors. Measuring the PDRs between nodes might take a long time, this is however an acceptable compromise considering the desired properties of the PDR approach.

The discovery stage using the PDR approach is simulated for a node in the middle of a 200 node network. The network is a circle with a radius $r \approx 33.8\,\mathrm{m}$, which is the same area as a 60 m by 60 m square network. The remaining nodes are distributed uniformly using MATLAB's `rand` function. Using a neighbourhood BLEP threshold = 0.95, the node has 98 actual neighbours, and there are 102 out of range nodes. Using $\texttt{PPS} = 0.5\,\mathrm{s}^{-1}$, the probability of a node beginning a colliding transmission q is calculated assuming nodes can break receipt by Equation (7.1).

$$q = \frac{\Delta t_{v,41\,\mathrm{B}}}{\frac{1}{\texttt{PPS}}} = \frac{0.656\,\mathrm{ms}}{\frac{1}{0.5\,\mathrm{s}^{-1}}} = 0.328 \cdot 10^{-3} \tag{7.1}$$

Where:

$\Delta t_{v,41\,\mathrm{B}}$	is the vulnerable period for a 41 B long packet, see Section 2.4.2.	[s]
PPS	is the injection rate.	$[\mathrm{s}^{-1}]$
q	is the probability that a node injects a colliding packet, when a node is transmitting on the same channel as the receiver is listening.	[1]

The PPS is chosen as $\texttt{PPS} = 0.5\,\mathrm{s}^{-1}$ since a high PPS results high average noise level. With a high average noise level, the range of the nodes is reduced (see Figure 7.1). Therefore a low PPS is desired to get a correct reading of the neighbourhood. This is under the assumption that the traffic is low during normal operation. If traffic during normal operation is high, setting up the network in more noise might result in a better performing network. Based on this, the value $\texttt{PPS} = 0.5\,\mathrm{s}^{-1}$ is chosen as it is a low value[5].

The quantity of nodes transmitting interfering packets is chosen randomly from a Poisson distribution with the probability q and 200 trials. Each node in the network has equal probability of transmitting. If a node is found to be transmitting a colliding packet, it is assumed that it is transmitting with a constant power throughout the duration of all other transmissions for the sake of simplicity[6]. From the positions of interfering nodes, the interference power is calculated. One packet from every node is simulated in the calculated interference. This is done n times for each point with new interference power levels. The PDRs are compared with a $\mathrm{PDR}_{\mathrm{cutoff}} = 0.9$, to determine whether the nodes are neighbours. Figure 7.3 shows the results of the simulation.

[5] Ideally the value of PPS should be chosen dynamically or based on some known values, so the implementation would work for a range of different network configurations. A static value of $\texttt{PPS} = 0.5\,\mathrm{s}^{-1}$ will however have to suffice for this project due to limited time.

[6] The receipt breaking behaviour and constant power levels throughout the whole transmission results in an increased BLEP compared to the network simulator. This should result in more type 1 errors and fewer type 2 errors in these calculations compared to what the network simulator will show.

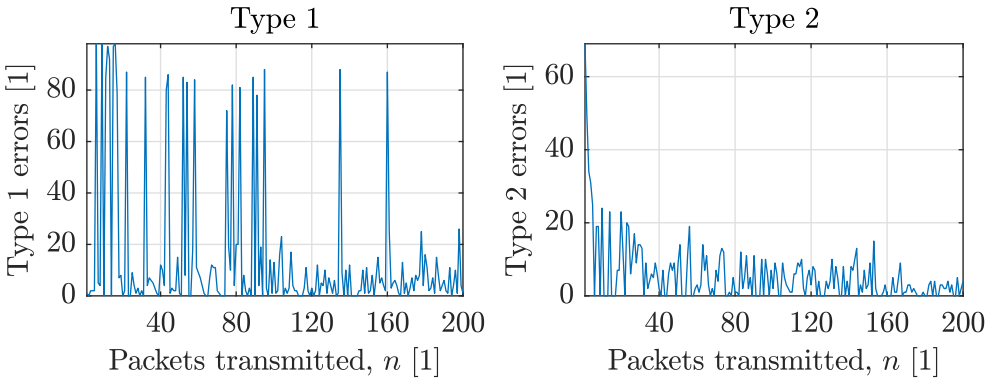


Figure 7.3: Data from rough simulation of neighbourhood discovery with $\text{PDR}_{\text{cutoff}} = 0.9$. There are 98 actual neighbours and 102 out of range nodes placed by the matlab `rand` function in a 60 m by 60 m area.

Figure 7.3 shows that there is a probability of making a lot of type 1 errors, as shown by the high spikes in the graph. These spikes happens due to a nearby node being chosen as interference, thus causing most packets to be lost. The tall spikes reaches values close to the real neighbour count of 98, meaning that most of the actual neighbours are deemed not neighbours. When this happens, the node will only accept a select few as its relay, causing redundant relays to be created. Figure 7.3 also shows that the node will make approximately 10 type 2 errors all the way up until 150 packets has been transmitted. Since type 2 errors can cause nodes to accept out of range relays, these errors should be avoided to ensure all nodes have a good connection to the rest of the network.

The $\text{PDR}_{\text{cutoff}}$ is therefore increased to $\text{PDR}_{\text{cutoff}} = 0.95$, and the data is shown in Figure 7.4.

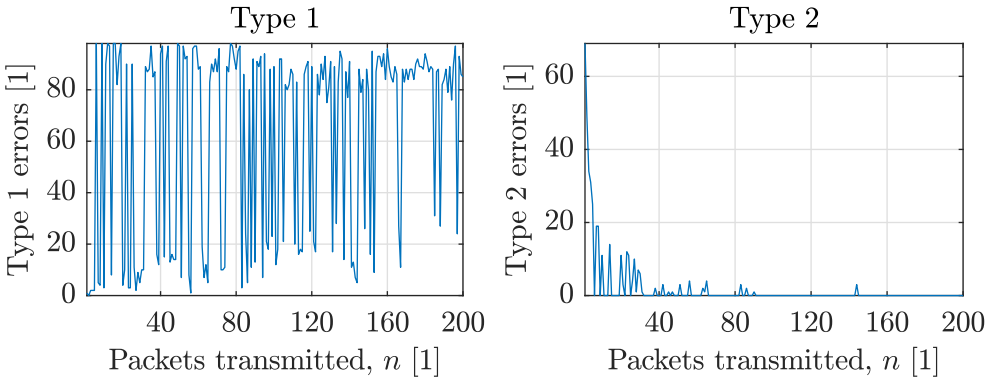


Figure 7.4: Data from rough simulation of neighbourhood discovery with $\text{PDR}_{\text{cutoff}} = 0.95$. There are 98 actual neighbours and 102 out of range nodes placed by the matlab `rand` function in a 60 m by 60 m area.

When $\text{PDR}_{\text{cutoff}} = 0.95$, the probability of the node making a lot of type 1 errors is significantly increased, meaning increased risk of redundant relay nodes. The quantity of type 2 errors is however reduced to a point where there is close to no errors at $n = 40$ packets. Since type 2 errors are of higher cost than type 1 errors, these results are deemed acceptable.

It is chosen that nodes should transmit at least $n = 40$ packets for the discovery stage. Nodes schedule their next transmissions uniform randomly within an interval $\left]0, \frac{2}{\texttt{PPS}}\right]$[7] after finishing their last transmission. The time that should be dedicated to discovery, to ensure that all nodes have transmitted atleast $n = 40$ packets, is therefore given by Equation (7.2).

$$\Delta t_d = n \cdot \left(\frac{2}{\texttt{PPS}} + 2 \cdot \Delta t_{CC} + 3 \cdot \Delta t_{\text{trans}} \right) \approx 161\,\text{s} \tag{7.2}$$

Where:

Δt_d	is the time frame dedicated to discovery	[s]
n	is the minimum quantity of packets chosen for discovery	[1]
Δt_{CC}	is the maximum time between channel change and transmission during an advertisement event. Arbitrarily chosen as $\Delta t_{CC} = 10\,\text{ms}$	[s]
Δt_{trans}	is the time maximum takes to transmit a packet. Max packet size $= 47\,\text{B} \implies \Delta t_{\text{trans}} \approx 0.38\,\text{ms}$	[s]

The network simulator is used to simulate the results of using $\text{PDR}_{\text{cutoff}} = 0.95$, $\texttt{PPS} = 0.5\,\text{s}^{-1}$, and $\Delta t_d = 161\,\text{s}$ for a 200 node 60 m by 60 m square area network. The PDRs are calculated using the `NeighbourMatrix`es of the nodes after simulating $t = 161\,\text{s}$. The method of calculating the PDRs are given by Equation (7.3).

$$\text{PDR}_i = \frac{\text{NM}(i, 11)}{\text{NM}(i, 9)} \quad [1] \tag{7.3}$$

Where:

PDR_i	is the PDR of the i-th node in the `obj.NeighbourMatrix`	[1]
$\text{NM}(i, 11)$	is the $(i,11)$ entry of the `obj.NeighbourMatrix`, which contains the amount of packets received	[1]
$\text{NM}(i, 9)$	is the $(i,9)$ entry of the `obj.NeighbourMatrix`, which contains the amount of packets the sender has send	[1]

Figure 7.5 shows the PDRs from the `NeighbourMatrix`es for all nodes, in relation to the distances between the nodes.

[7] A node cannot send a message right after receiving, there is a small delay.

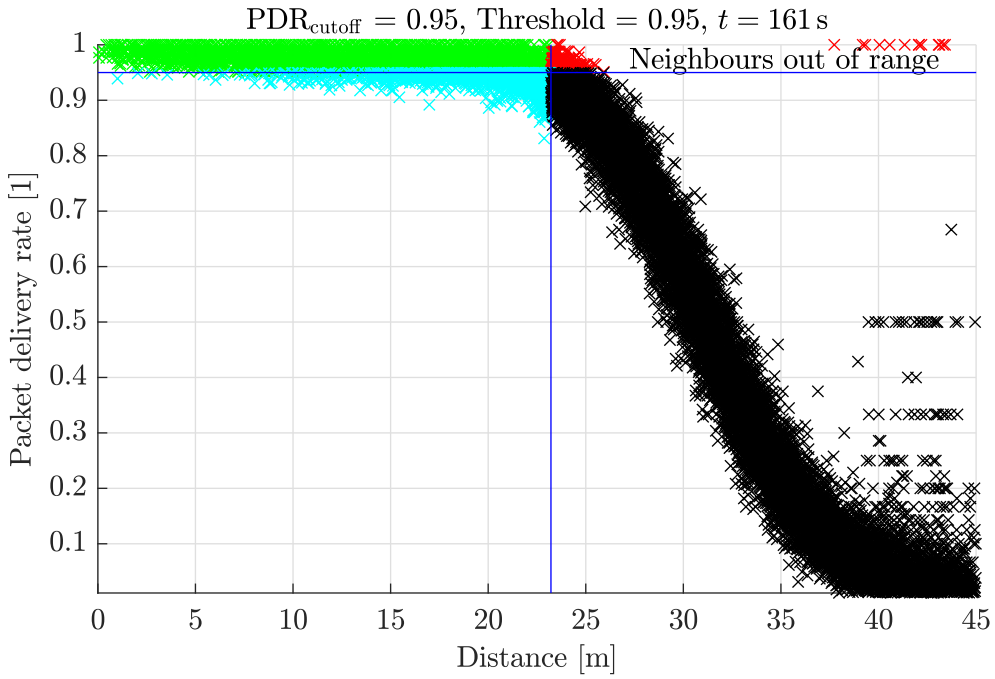


Figure 7.5: Data from the network simulator. Plot of connections between nodes after simulating 160 s. Every cross denotes a connection between two nodes. Red crosses denote type 2 errors, and cyan crosses denotes type 1 errors. There is a total of 233 type 2 errors, and 1376 type 1 errors.

Figure 7.5 shows that most of the out of range nodes has been detected correctly, and that nodes on average makes $\frac{1376}{200} \approx 6.9$ type 1 errors and $\frac{233}{200} \approx 1.2$ type 2 errors. The figure however also shows that some type 2 errors are made between nodes with distances up to 43.4 m, which means that these nodes can accept a relay where transmissions have a BLEP up to 0.996 for packets of 47 B, assuming no noise (see Figure 7.1). This can happen if a node u has successfully received the first packet from a distant node v. This will result in the node u adding v's information to its `NeighbourMatrix`. If u never receives another packet from v, v's entry in u's `NeighbourMatrix` will not be updated. This means that the (v,11) and (v,9) entries will remain equal one, resulting in a $\text{PDR}_v = 1$.

These errors can be avoided by defining a threshold R_{min} denoting the minimum quantity of packets that a node should have received from another node before it is a potential neighbour. The discovery stage runs for a pre-defined time period to ensure that all nodes have transmitted at least $n = 40$ packets. Since a PDR threshold of $\text{PDR}_{\text{cutoff}} = 0.95$ is used, the minimum number of packets a node should have received from a neighbour is calculated using Equation (7.4).

$$R_{\text{min}} = \left\lfloor \frac{\Delta t_d \cdot \texttt{PPS}}{2} \cdot \text{PDR}_{\text{cutoff}} \right\rfloor = \lfloor\, n \cdot \text{PDR}_{\text{cutoff}} \rfloor = 38 \tag{7.4}$$

Where:

R_{min}	is the minimum quantity of packets a node should have received from a neighbour	[1]
n	is the minimum quantity of packets chosen for discovery	[1]

`PPS`	is the packet injection rate for a node	$[s^{-1}]$
Δt_d	is the duration of the discovery stage	[s]
PDR_{cutoff}	is the PDR threshold for when a node is considered a neighbour	[1]

Using the same data from Figure 7.5, the decisions are plotted again after removing all entries where `NeighbourMatrix`$(i, 11) < 38$ in Figure 7.6.

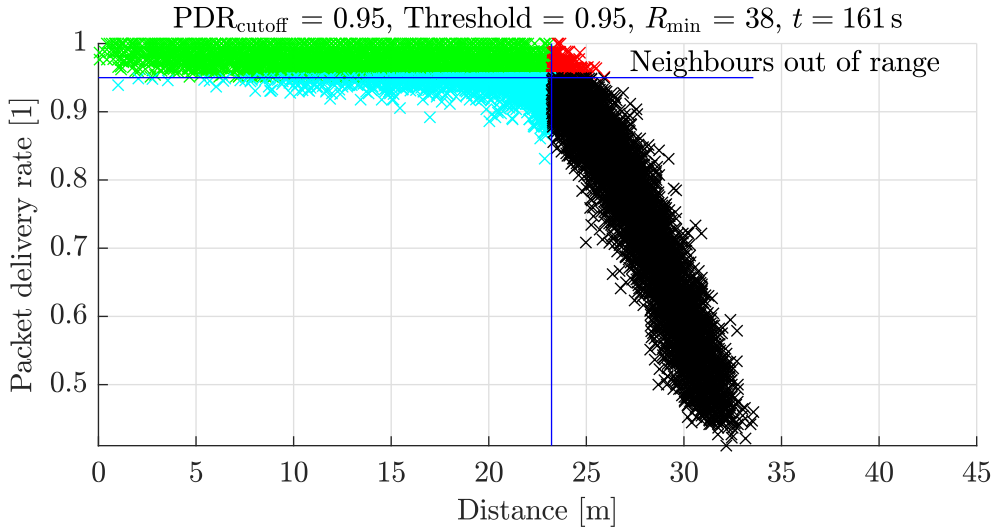


Figure 7.6: Data from the network simulator. Plot of simulated connections between nodes after simulating 160 s. Connections with less than R_{min} packets received are not displayed. Every cross denotes a connection between two nodes. Red crosses denote type 2 errors, and cyan crosses denotes type 1 errors. There is a total of 218 type 2 errors, and 1376 type 1 errors.

After implementing R_{min}, the average node makes $\frac{1376}{200} \approx 6.9$ type 1 errors and $\frac{218}{200} \approx 1.1$ type 2 errors. Figure 7.6 shows that type 2 errors are made between nodes with distances up to approximately 26 m, which means that these nodes can accept a relay where transmissions have a BLEP of up to 0.16 for packets of 47 B assuming no noise (see Figure 7.1). These are significantly better results than without R_{min}, where transmissions has a BLEP of up to 0.996.

The network simulator is run for a longer period, and the quantity of errors is plotted in respect to time on Figure 7.7.

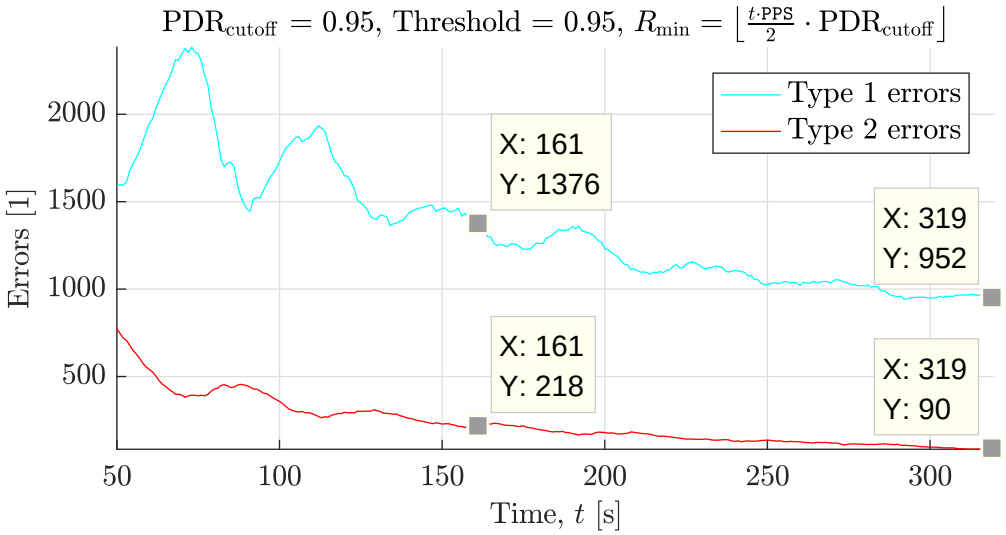


Figure 7.7: Data from the network simulator. Plot of simulated decision errors using PDR$_{\text{cutoff}} = 0.95$ in respect to time. Dedicating more time to discovery, results in less errors.

In Figure 7.7 there is ripple, this can be explained by looking into the binomial distribution. As the quantity of received packets and packets sent are integers, and the PDR is compared with a static number, the probability of getting a PDR lower than the threshold varies with the quantity of packets sent. Consider the two cases where 60 and 79 packets have been send. If four packets are lost in both these cases, the PDR will be lower than 0.95. Using a binomial distribution [19] with a probability of packet loss of $p = 0.05$, the probability of losing more than three packets are given in Equations (7.5) and (7.6).

$$Pr(X > 3)_n = 1 - \sum_{k=0}^{3} \frac{n!}{(n-k)! \cdot k!} \cdot p^k \cdot (1-p)^{n-k}$$

$$Pr(X > 3)_{60} = 1 - \sum_{k=0}^{3} \frac{60!}{(60-k)! \cdot k!} \cdot p^k \cdot (1-p)^{60-k} \approx 0.35 \tag{7.5}$$

$$Pr(X > 3)_{79} = 1 - \sum_{k=0}^{3} \frac{79!}{(79-k)! \cdot k!} \cdot p^k \cdot (1-p)^{79-k} \approx 0.56 \tag{7.6}$$

Where:

X	is a binomial random variable denoting the quantity of lost packets	[1]
p	is the packet loss probability	[1]
n	is the quantity of transmitted packets	[1]

As seen from Equations (7.5) and (7.6), the probability of having a PDR lower than 0.95 varies with the quantity of transmitted packets. This observation explains the ripples in Figure 7.7.

Figure 7.7 shows that dedicating more time to discovery decreases the quantity of errors until it converges at a point. When long time is dedicated to discovery, the average transmission encounters a small interference, increasing the BLEP slightly. Therefore the

BLEPs between nodes should converge to a slightly higher number than in the no-noise case. Since errors are determined by comparing with the zero-noise range, the quantity of type 2 errors should converge to 0, and the type 1 errors should converge to a value > 0.

Figure 7.8 shows the PDRs from the `NeighbourMatrix`es for all nodes, in relation to the distances between the nodes after simulating $t = 319\,\text{s}$.

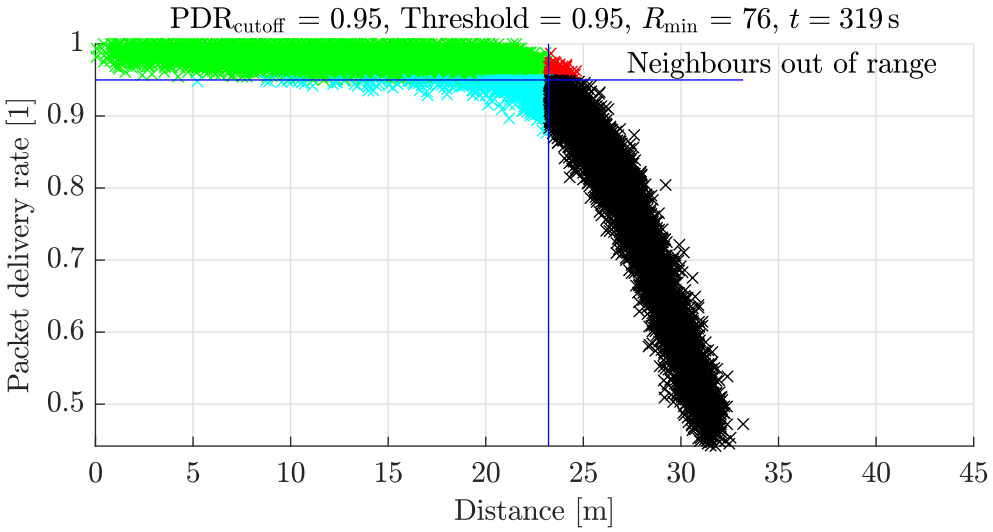


Figure 7.8: Data from the network simulator. Plot of simulated connections between nodes after simulating 319 s. Connections with less than R_{min} packets received are not displayed. Every cross denotes a connection between two nodes. Red crosses denote type 2 errors, and cyan crosses denotes type 1 errors. There is a total of 90 type 2 errors, and 952 type 1 errors.

In this case the average node makes $\frac{952}{200} \approx 4.8$ type 1 errors and $\frac{90}{200} = 0.45$ type 2 errors. Figure 7.6 shows that type 2 errors are made between nodes with distances up to approximately 24.6 m, which means that these nodes can accept a relay where transmissions have a BLEP of up to 0.09 for packets of 47 B assuming no noise (see Figure 7.1).

Since it has been found that extending the discovery stage duration to $t = 319\,\text{s}$ does not offer significant reduction in neighbourhood discovery error quantity compared to $t = 161\,\text{s}$, it is assessed that a discovery duration of $\Delta t_d = 161\,\text{s}$ offers a good compromise between setup duration and neighbourhood discovery error quantity. It is therefore chosen to run the discovery stage with the variables listed in Table 7.4.

Table 7.4: Table listing the variables chosen for discovery in a 200 node 60 m by 60 m network.

Name	**Value**
PDR$_{\text{cutoff}}$	0.95
Δt_d	161 s
R_{min}	38

It should be noted that these values only hold true for a 200 node network in a 60 m by 60 m square area. Due to limited time, a general solution for different areas and node

quantities will not be derived.

When a node initialises discovery, it will set its `step` $= 0$.

After a node has been in the discovery stage for Δt_d it will determine what nodes are neighbours and save this information. The node will change its `step` to 1, which will be transmitted in its next setup packet. If a node receives a packet where `step` $= 1$ and has `step` $= 0$, it will determine what nodes are neighbours based on its `NeighbourMatrix` and change its `step` to 1. This way, when one node has completed the discovery stage, it will be propagated throughout the network.

With the discovery stage completed, the relay distribution stage will be discussed and implemented.

7.5 Stage 2: Relay distribution

In the Relay distribution stage nodes broadcast packets periodically in order to determine which nodes are to become the relays of the network and share information. The relay distribution stage will follow the overall structure of the highest connectivity based greedy algorithm described in Section 6.1.

In the relay distribution stage nodes will be in one of four `Step`. The `Step` of a node indicates how far along the configuration process the node is. A node in `Step` 1 will decide if the node itself or a neighbour node will become a relay, this decision process is described in Section 6.1 as the construction phase.

A node in `Step` 2 is about to decide itself or itself and a neighbour should become relays to connect the network, this process is described in Section 6.1 as the connection phase.

`Step` 3 means that the node has finished its stage of the network configuration. The node will continuously check if all its neighbours are finished as well, before the node can advances to `Step` 4.

A node is in `Step` 4 if it has confirmed, that everyone in its neighbourhood and neighbours neighbours are finished with the network configuration.

One difference between the implemented algorithm and the theoretical algorithm described in Section 6.1 is the case where two or more nodes have the same amount of undiscovered neighbours in `Step` 1. In this event it is chosen that the node with the lowest ID is chosen as relay, this increases the probability of two nodes pointing at the same node as relay, in the event of simultaneous decisions.

Nodes broadcast packets periodically. In order to determine a value of `Packets Per Second (PPS)` an analysis is made. Appendix G contains the data points, curves, and MATLAB code used. An expression for an optimised `PPS` dependent on the amount of neighbours is desired.

$10 \cdot 100 = 1000$ simulations are run, these simulations have varying number of nodes and different `PPS` settings. All nodes are placed in a small area making it possible for every

node to see all nodes, to simulate a neighbourhood. The simulation results in 10 sets of data points, which describes the average Packet Delivery Ratio (PDR) dependent on the amount of nodes in the network and the PPS setting.

Fifth degree polynomial curves are fitted to these 10 sets of PDR data points. This results in 10 individual functions PDR(PPS) for different neighbourhoods. The 10 curves are used to solve the minimization problem seen in Equation (7.7), for different node quantities.

$$\begin{aligned} \text{minimize:} \quad & \text{cost} = \frac{\text{PBD}}{\texttt{PPS}} && [\text{s}] \quad (7.7) \\ \text{subject to:} \quad & \left(1 - (1 - \texttt{PDR})^{\text{PBD}})\right) \geq 95\% \\ & 0.1 \leq \texttt{PPS} \leq 10 \\ & \text{PBD} \in \mathbb{N} \\ & \text{PBD} \geq 1 \end{aligned}$$

Where:

PDR	is the Packet Delivery Ratio.	[1]
PBD	is the is the amount of packets a node needs to send before neighbours have a 95% probability of receiving at least one packet.	[1]
PPS	is the is the amount of packets sent per second.	$[\text{s}^{-1}]$
cost	is the time it takes to get to 95% certainty that at least one packet has arrived.	[s]

The cost of the function seen in Equation (7.7) is time. It is chosen to minimize time because the configuration process is desired to be fast. Packets Before Decision (PBD) describes how many packets a node needs to send before there is a 95% probability that a neighbour has received at least one packet. PPS describes how many packets a node sends every second.

When the cost function seen in Equation (7.7) is minimised, an optimised PPS is found for different amount of nodes. A single curve is fitted to these data points and a function for the optimal PPS, dependent on the amount of nodes that can interfere with a node's communication. The expressions is given by Equation (7.8).

$$\texttt{PPS} = 11.3405 \cdot \text{e}^{-0.0017334 \cdot \texttt{nodes}} \qquad [\text{s}^{-1}] \quad (7.8)$$

If a PPS outside the interval $0.1 < \texttt{PPS} < 10$ then the nearest value in the interval is chosen.

In the case of a 60 m by 60 m network, a node is at most 84.85 m away. Figure 7.1 shows that a nodes at 100 m will still interfere with communication within a node's neighbourhood. The amount of nodes that can interfere with a node's communication is therefore chosen as the total amount of nodes, 200.

Nodes will wait on each other and make decisions one at a time. The node at the lowest step will make a decision first. In the case of more nodes at the same step the node with the lowest ID will make a decision first. Whenever a node is next to make a decision it will set a variable called TTD (Time to decision).

If a node is in `Step` 2 and it receives a packet from a neighbour, where the `CID` has changed the node will prolong its `TTD`. This is done to allow the `CID`s to propagate throughout the network before making decisions.

A list called `waitList` will be kept at each node. The list contains the ID of all the trusted neighbour nodes who have marked another node to become a relay. The `waitList` list is updated every time a packet is received. A node will reset its `TTD` as long as the `waitList` is not empty.

`TTD` is a point in time when the next decision is made.If node u just made a decision, a node v needs to allow for node u's decision to propagate in the network, before node v can make a decision. The purpose of `TTD` is to ensure that the previous decision propagates before the next decision is made.

Before the actual value of `TTD` is found, the importance of the delay is explained by analysing the node setup seen on Figure 7.9.

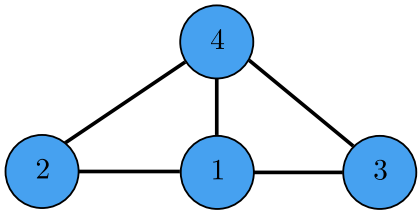


Figure 7.9: A simple 4 node setup used to illustrate the importance of the implemented `TTD` variable.

In the setup seen on Figure 7.9 all nodes are in `Step` 1. Node 1 will make the first decision. Node 2 and 3 will both wait on node 1 before they can make their decisions. When node 1 makes a decision it will mark itself as a relay.

If there is no delay before node 2 and 3 make their decisions they will both mark themselves as relays. Because they both see one unconnected neighbour, node 4, even though node 4 actually is dominated.

If a delay is implemented node 4 will have a chance to inform node 2 and 3 that it is dominated, before they make their decisions. Thus the delay ensures that only node 1 becomes a relay node.

Whenever a node sets the `TTD` three calculations are needed Equation (7.9), Equation (7.10), and Equation (7.11).

In Equation (7.9) the worst PDR of all the nodes in the trusted neighbours is found.

In Equation (7.10) the expression $\left\lceil \frac{-3}{\log(1-\texttt{PDR}))} \right\rceil$ is derived from the Cumulative distribution function (CDF) of a geometric random variable, $\left(1-(1-\texttt{PDR})^{\texttt{PBD}}\right)$. When the expression is set equal to 0.95, `PBA` describes the number of packets a node is required to send before its neighbours have a 95% chance of receiving at least one of the packets. 95% is chosen because it is assessed to give a fair compromise between cost and probability of successful transmissions. `PBA` must be an integer value therefore it is rounded up. This is done since

the nodes can not transmit a fraction of a packet, this choice will further improve the probability of successfully reaching a neighbour node.

If the value of PBA is found to be less than 3, it is set to 3. This is done since Appendix G has found that the PBA should be equal 3 in most cases. So if the PDR is read incorrectly, leading to a too low PBA, the PBA be set to a value that has been found to work in the neighbourhood simulations.

In Equation (7.11) the TTD value is set. The value is calculated based on how much time a node maximally can use to send PBA amount of packets. When the TTD value is set, a decision has been scheduled.

$$\mathtt{PDR} = \min_i \left(\frac{\mathtt{NM}(1:\mathtt{numTN}, 11)_i}{\mathtt{NM}(1:\mathtt{numTN}, 9)_i} \right) \quad [1] \quad (7.9)$$

$$\mathtt{PBA} = \max \left(\left[\left\lceil \frac{-3}{\log(1-\mathtt{PDR}))} \right\rceil, 3 \right] \right) \quad [1] \quad (7.10)$$

$$\mathtt{TTD} = \mathtt{time} + \left(2 \cdot \frac{1}{\mathtt{PPS}} \cdot \mathtt{PBA} + 2 \cdot \Delta t_{CC} \cdot \mathtt{PBA} + 3 \cdot \mathtt{PBA} \cdot \Delta t_{\text{trans}} \right) \quad [s] \quad (7.11)$$

Where:

PDR	is the calculated minimum PDR of the trusted neighbours.	[1]
PBD	is the is the amount of packets a node needs to send before neighbours have a 95% probability of receiving at least one packet.	[1]
numTN	is the amount of trusted neighbours.	[1]
NM(j,11)	is the (j,11) entry in the obj.NeighbourMatrix, which contains the amount of packets received.	[1]
NM(j,12)	is the (j,12) entry in the obj.NeighbourMatrix, which contains the amount of packets the sender have tried to send.	[1]
PBA	is expected packet transmissions needed to reach a neighbour.	[1]
TTD	is a simulation variable, which describes at what time a decision should be made.	[s]
time	is the simulation variable containing the current time.	[s]
Δt_{CC}	is the simulation variable, maxRndWaitChangeChannel.	[B]
Δt_{trans}	is the simulation variable, which contains the total packet length.	[B]

If the example seen on Figure 7.9 is again considered. It is seen that with the chosen implementation there is a 5% chance that node 2 will not heave received a packet from node 4, before node 2 makes its decision. This is because PBA is calculated from a 95% probability of delivery threshold. A higher threshold could have been chosen. However an increase of the threshold would cause an increase of the PBA variable. This would cause a longer delay between decisions contributing to a longer network configuration time.

7.5.1 Configuration time estimation

The time the relay distribution stage of the configuration process requires can be estimated by Equation (7.12). This estimate is a rough estimate where all nodes in the network are within range.

When all nodes are within range only one relay is chosen and all other nodes is marker as dominated and have to check if they are able to connect any unconnected relays.

In practice not only one node is marked as a relay. Since nodes would not have all other nodes in the network within their range. Thus nodes would only have to wait on other nodes in their neighbourhood. If a node becomes a relay it will move to `Step` 3 and not make any more decisions. If a undiscovered node is marked as dominated node it will skip the decision of `Step` 1. Thus, it is estimated that each node will make one decision.

$$\text{Estimated time} = \texttt{Neighbours} \cdot \frac{\texttt{TTD} - \texttt{time}}{\texttt{oneSec}} \tag{7.12}$$

Where:

Estimated time	is the estimated time	[s]
`TTD`	is the time delay between nodes making decisions in simulation ticks	[1]
`Neighbours`	is the amount of nodes in the network	[1]
`oneSec`	is the simulation ticks per second	$[s^{-1}]$

If a PDR of 40% is assumed for all nodes in a 200 node setup. The estimated time calculated by Equation (7.12) becomes 324 s, since TTD is calculated as 1.6 s.

In Equation (7.11), *`TTD` is stated to be* $... + 2 \cdot \Delta t_{CC} \cdot$ *`PBA`* $+ ....$ *However a mistake was made, in the code it was implemented as* $... + 2 \cdot \Delta t_{CC} \cdot 40 + ...$ *This mistake will mean an increased delay between nodes making decisions. This should result in overall slightly better PDR but a slower configuration time for networks with high PDRs. With a higher PDR the network is less likely to make "wrong" decisions that will result in fewer relays and less redundancy. The estimated required time with the wrong implementation is found to be* 460 s.

With the design of the implemented algorithm described, a validation of the resulting algorithm is now be made. The MATLAB code which runs on the nodes in the network simulator to implement the above, is given by Appendix N. The validation is done in Chapter 8 and compares the theoretical from Section 6.1 and the implementation in this chapter.

Acceptance simulation 8

The performance metrics from Chapter 3 are used to compare the implemented solution from Chapter 7 to the theoretical case described in Section 6.1.

The metrics that are used to compare the two cases are given in Chapter 4. This chapter starts with a short introduction to the testing methodology and a comparison of the number of relays, which gives insight into the volatility of the implementation. The full methodology is found in Appendix F where scripts that run the simulations is found.

Afterwards each metric has its own section wherein the point of the metric is refreshed. Most metrics are represented with a graph that shows how it changes across traffic intensity. At the end of each section, a conclusion is drawn from the data, leading to the discussion and conclusion of the report.

8.1 Simulation methodology

The point of this chapter is to show with a high degree of confidence, if the implemented algorithm performs equivalent to the theoretical. To do this, 24 random number generator seeds are chosen. These seeds are given in the first column of Table 8.1. The reason why the first 12 seeds are irregular, is because they need to satisfy the condition that a 1000 node network is connectible in a 330 m by 330 m area, Section 4.2. The seeds are therefore chosen by trial and error.

To show how the algorithms handles different amounts of traffic, the networks created by the algorithms are simulated at different traffic level intensities. Since rapid changes in performance is expected at lower traffic intensities, the traffic intensity resolution is higher at lower levels. This is done by choosing the traffic intensities by the MATLAB command: `round(logspace(log(5),log(200),12))`. This command gives logarithmically spaced numbers from 5 to 200. In each test a traffic level and a seed is simulated until the nodes have tried to send 1000 packets in total.

Table 8.1: Table listing the variables used for the simulation and the results from the simulations.

Nodegen	**Time [s]**	**Relays (implementation)**	**Relays (theoretical)**
200	324	14	8
230	334	9	11
260	314	13	11
290	305	11	10
320	333	8	9
360	344	13	12
380	330	11	10
410	342	8	9
460	332	10	12
470	325	11	10
500	334	10	9
540	336	11	10
560	337	12	11
580	326	12	8
600	314	10	10
620	340	9	8
640	332	12	8
660	326	8	11
680	361	10	9
700	323	15	10
720	309	14	10
740	330	11	8
760	338	11	8
780	338	11	10

In Table 8.1 the column **Nodegen**, specifies which seed is used for generating the node set. The next column **Time**, is how long the implemented algorithm takes to assign relay functionality. The third column **Relays (implementation)**, gives the number of relays that were made by the implementation. The fourth column **Relays (theoretical)**, is the amount of relays that the theoretical implementation made.

Table 8.2: Sample mean and variance for the number of relays of the theoretical and the implementation.

Name	**Relay Sample mean**	**Unbiased sample variance**
Theoretical	9.67	1.62
Implementation	11	3.65

In Table 8.2, sample mean and variance for the two cases are seen. It is seen that the theoretical case gives noticeably fewer relays, suggesting that the implementation is worse. This is further suggested by the sample variance which shows that the implementation is more volatile. In the next section the configuration time is given from Table 8.1.

8.2 Configuration time

The first performance metrics to test is the Configuration time. This measures how long time the algorithm requires to configure the network, with different node sets. The resulting times can be seen in the following table Table 8.1.

The average time for setup is 330.3 s. The time given for the discovery of neighbours is 161 s, which means that on average the network took 169 s to distribute relays after neighbourhood discovery.

In Section 7.5.1 the configuration was estimated to be 460 s. However it is seen that the configuration process on average require 37% of the estimate time. This can be due to the fact that more nodes can make simultaneous non-conflicting decisions when all nodes are not within range of each other.

8.3 Packet delivery ratio

An indicator for how well a network performs is the Packet Delivery Ratio (PDR). This is calculated by summing all the successfully received packets and dividing with all the attempts at sending packets. The PDR is calculated for each node set and averaged for each traffic level. This produced the graph seen on Figure 8.1.

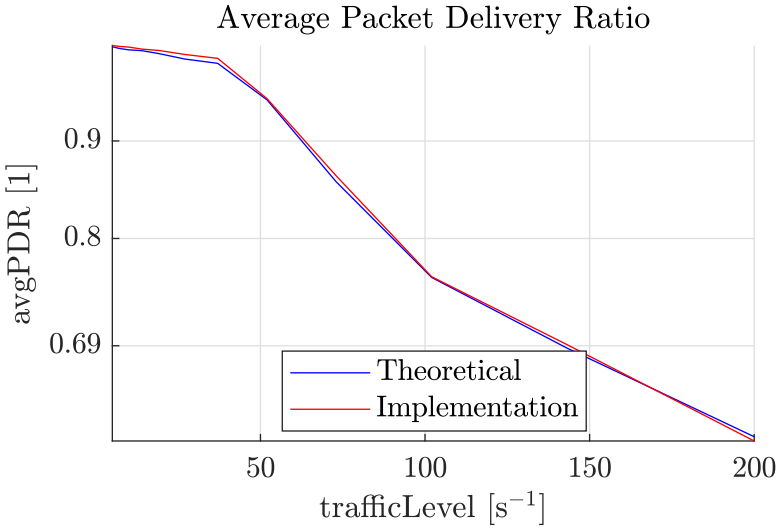


Figure 8.1: The average PDR for diffrent traffic levels

It is seen in Figure 8.1 that the theoretical and implementation is nearly identical. This is good since it shows that the implementation performs as well as the theoretical. For added rigour, a statistical test is done at each traffic level. Equation (8.1) is a test of equality

[20] which is approximately correct when n is large.

$$z = \frac{\hat{p_1} - \hat{p_2}}{\sqrt{\hat{p}(1-\hat{p})\left(\frac{1}{n}+\frac{1}{n}\right)}} \tag{8.1}$$
$$\hat{p} = \frac{\hat{p_1} + \hat{p_2}}{2}$$

Where:

$\hat{p}$	is average PDR.	[1]
$\hat{p_1}$	is the first PDR.	[1]
$\hat{p_2}$	is the second PDR.	[1]
n	is the number of tests, which is 24.	[1]
z	is approximately a standard normal random variable.	[1]

The test compares the H0 hypothesis that the two PDRs are equal to each other with the H1 hypothesis that they are not equal to each other. The z value is calculated from Equation (8.1) and compared with $z_{\frac{\alpha}{2}}$ where α is the significance level. $z_{\frac{\alpha}{2}}$ is calculated using the MATLAB command `norminv(1-significance_level/2,0,1)`. At the 5 percent significance level, no value is found to be significantly different from each other.

8.4 Data throughput

Throughput is measured in kB s^{-1}, which gives a measurement of how much data is getting through the network. The result is seen on Figure 8.2.

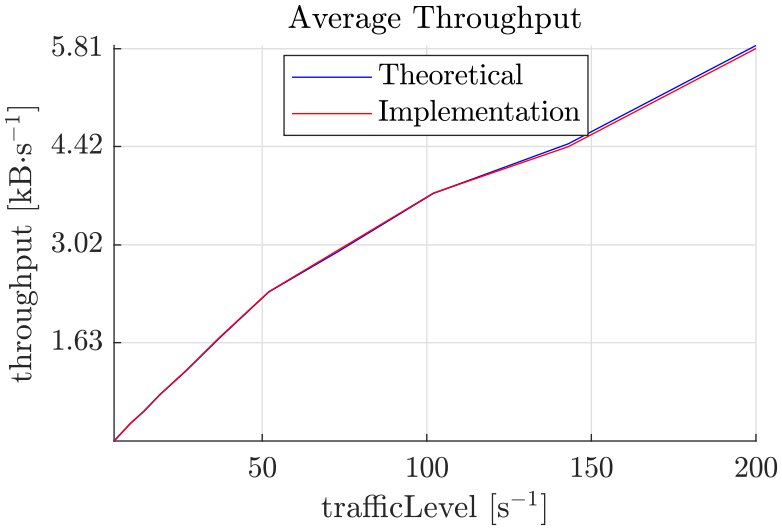


Figure 8.2: The average throughput for different traffic levels.

The average throughput of the algorithms is nearly identical.

8.5 Latency

Latency is calculated as the time it takes to get from the source node to the destination node. The time for each packet to arrive is averaged over all packets. The average latency for a packet that does not arrive is not calculated. The Average latency across tests is given in Figure 8.3, in which it is seen that the implementation lags behind the theoretical.

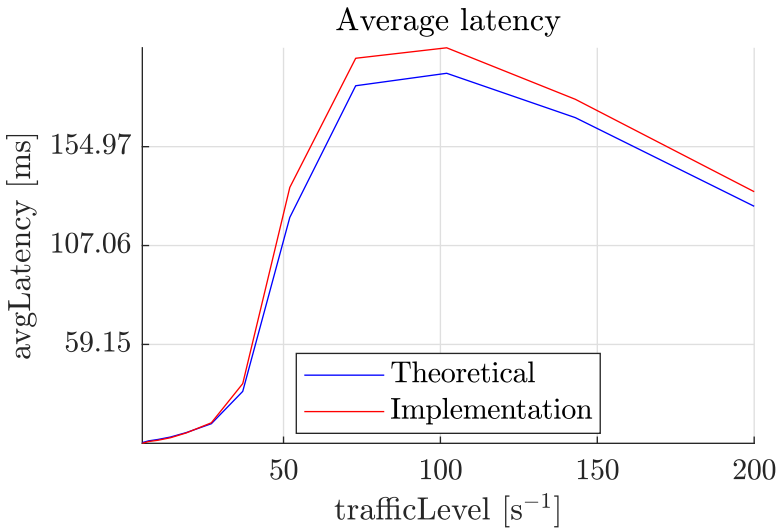


Figure 8.3: The average latency depending on the traffic level.

This observed increase in latency might be caused by many different parameters. However one possible explanation is that the average number of hops using the implemented algorithm is slightly higher then that of the theoretical, as seen on Figure 8.4. The same number of packets are delivered in both cases but one have more relays and uses more hops to deliver the packet. If more hops are used, the packet is transmitted more times, which could cause a increase in latency.

8.5.1 Average hops to destination

The average amount of hops to destination is defined as the quantity of nodes a packet has been relayed from before arriving to its destination. This means that a packet with a hop of 1 is from a neighbour. The average hops across experiments is plotted in Figure 8.4 over traffic level.

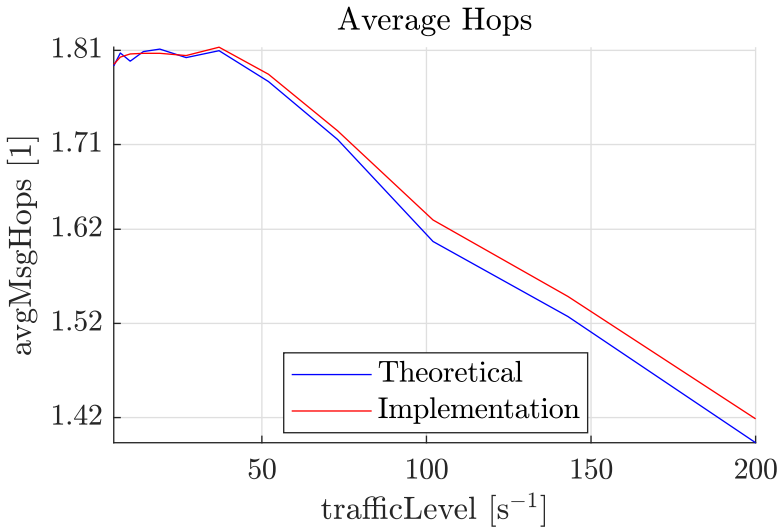


Figure 8.4: Plot showing the average hops the two cases utilises to deliver all successful packets.

It is seen on Figure 8.4 that both the theoretical and the implementation have ≈ 1.81 hops until the traffic level reaches 37 packet per second. After this point, it is seen that the implemented algorithm utilises more hops in order to deliver the packets. Interestingly this does not affect the Packet Delivery Ratio (PDR), so whether or not this makes an actual difference is hard to determine.

8.6 Fairness

Fairness is measured with three different methods: by Raj fairness using Equation (4.2), by minmax fairness using Equation (4.3), and by creating a histogram that shows how the Packet Delivery Ratio (PDR) is distributed.

Minmax fairness is plotted as an average over experiments across different traffic levels in Figure 8.5. The corresponding graph for Raj fairness is plotted in Figure 8.6.

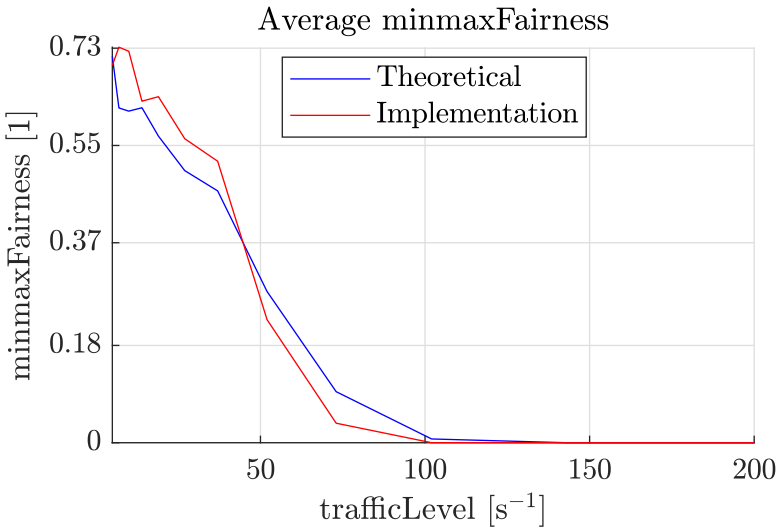


Figure 8.5: Graph of fairness calculation using Equation (4.3)

From Figure 8.5 it is seen that the minmax fairness indicator is more sporadic than the other metrics. Since there is not a big difference in the two algorithms, the two are seen as equal.

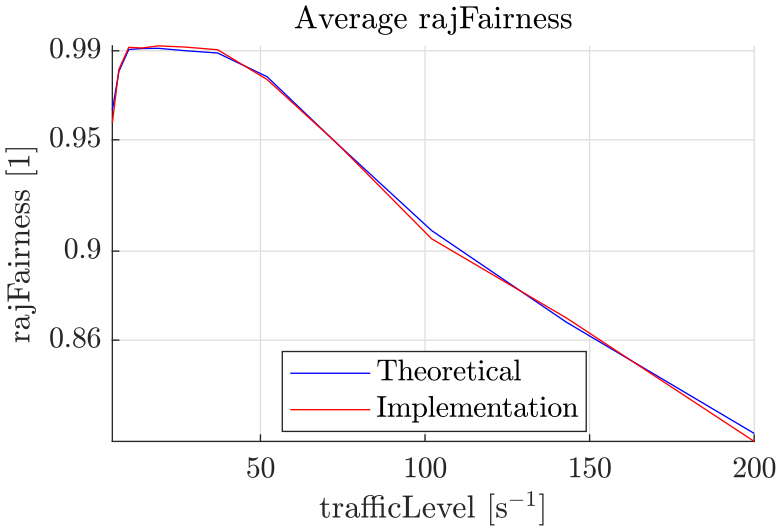


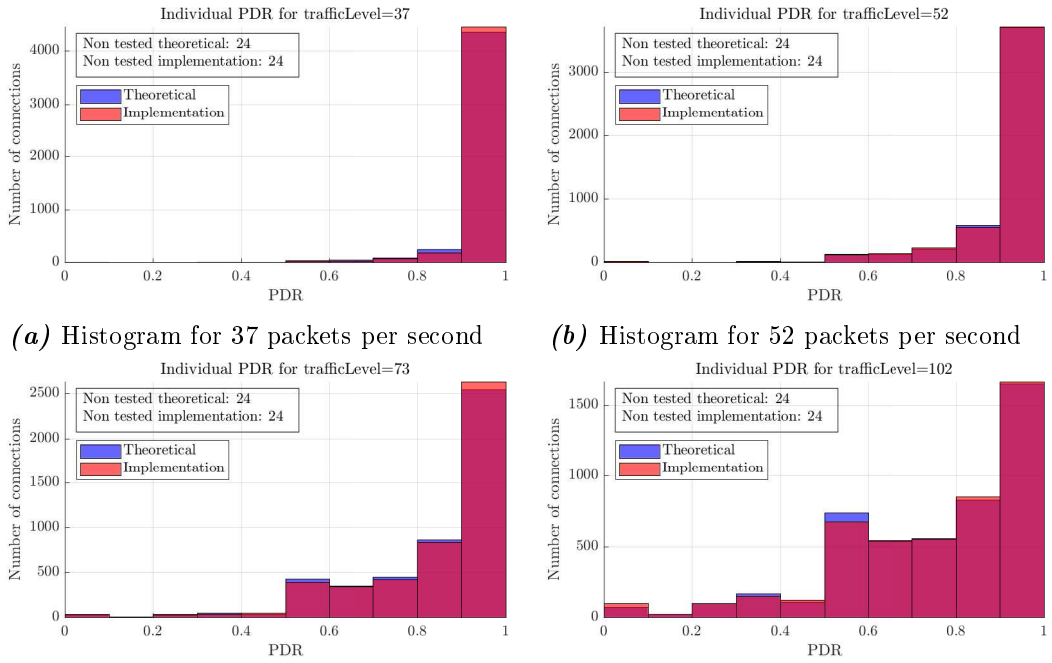
Figure 8.6: Graph of fairness calculation using Equation (4.2)

The graph of Raj fairness in Figure 8.6 shows the two algorithms to be almost equal.

To create histograms that shows the distribution of the PDRs, individual PDRs are calculated. This is calculated for each node that has sent one or more packets, by checking how many packets arrives at its destination. This is then gathered in a vector with values `NaN` when no packets have been sent, and a value from 0 to 1 indicating a PDR from 0 to 100%.

To get an average across experiments, each vector for a given traffic level intensity is concatenated. For each set of vectors, one for each algorithm, the histogram is plotted with PDR on the x-axis and the amount of nodes on the y-axis. Because of time constraints, not all nodes get to send packets. These nodes are not plotted since nothing can be said about them. This does give a degree of uncertainty regarding the fairness of the nodes not included.

Plots for each traffic level tested is in Appendix I. Some of the plots are given in Figure 8.7, where blue denotes theoretical and red denotes implementation. When the histogram is bordeaux, the two algorithms overlap.



(a) Histogram for 37 packets per second

(b) Histogram for 52 packets per second

(c) Histogram for 73 packets per second

(d) Histogram for 102 packets per second

Figure 8.7: Selected histograms for traffic level intensities 37 to 102.

The histograms in Figure 8.7 shows that most of the nodes have a PDR of close to 1. The two algorithms are very close to being equal in terms of distribution. As the traffic level goes from 37 to 102, the fairness decreases.

Discussion 9

In Chapter 8 the simulation of the implemented algorithm showed it to perform statistically equal to the theoretical case when looking at Packet Delivery Ratio (PDR). The simulation also showed that the implementation had, on average, 2 relays more than the theoretical case. This did not affect performance. The implementation also had a larger sample variance, as this did not affect the performance, it suggests that the performance metrics chosen are not overly sensitive to the number of relays.

Some challenges not considered in this project would have given the final solution more meaning in the real world. Some of these problems are given in this discussion.

The simulator uses the Additive White Gaussian Noise (AWGN) channel which is a channel model that results in no distortion of the signal. In the real world, signals undergo what is called fading which can make the signal harder to decode. By measuring how well the Bluetooth devices perform, a more realistic simulator could be made and the result of simulations would have more meaning.

The network setup does not consider which kind, and how much, noise the network will experience while functioning. To combat this, nodes could store which nodes they get messages from whilst normal communication is taking place. When sufficient information has been gathered, the network could reconfigure itself knowing which nodes can be reached with real noise.

The proposed solution assumes that all nodes who can be relay are stationary.In reality nodes are not either moving or stationary. An example is a microwave oven which is often stationary for longer periods, but can be moved to another location. Certain devices will most likely never be moved though, such as ceiling lamps, which could have a stationary feature. If only nodes with the stationary feature on can be relays, then the probability disconnecting the network, due to a crucial relay node being moved, would be low.

Even if only 100% stationary nodes are picked as relay nodes, breakdowns can happen. The first thing that can be done to avoid this, is to create multiple paths to all nodes. This would work most of the time, however there might be times where a higher degree of confidence in connectivity is needed. If a network has been disconnected by the loss of a crucial relay node, it could be reconnected through a two step approach: error registration and reconnection.

Error registration is that a node finds that one of its neighbouring relays have disappeared and reconnection is when the nodes reconnect themselves. Relays can disappear from a node's neighbour list. Error registration should be done such that nodes do not detect false positives, causing extra relay nodes to pop up even though the network is still

connected. Reconnecting the network after error registration should be done quickly without interfering with normal traffic.

To do this, nodes could keep a list of relays and a counter for each. When a relay has not been heard from in a reasonable time the node should send out a panic packet to start reconnecting the network.

Reconnecting can be done in many ways, one way could be to make nodes reset their relay ID and then have non-relay nodes connect the set as in the connection phase of Section 6.1.

It was chosen to use the algorithm HighConnect, but this has some downsides in sparse networks. If the network is sparse, meaning not many nodes, then the algorithm k2pruning would be more beneficial. From Section 6.5.1, k2pruning uses less packets for setup, and in Section 6.5.4 it is seen that k2pruning works noticeably better when nodes are very scattered. If nodes know the density of the network, which is assumed in Chapter 7, then the nodes could decide to either use one or the other.

Another reason why k2pruning is an interesting choice is its redundancy. It is easily seen in Appendix D, that k2pruning is much more interconnected than HighConnect. If a relay node disappears from the network, then there is a very low risk that any nodes will become unconnected. Thus, if redundancy is the most important metric, then k2pruning would be a better candidate than HighConnect.

In Section 4.1 the solution proposal evaluated two different methods of distributing relays, either it could be distributed or it could be central.

A privileged node could take care of the computationally heavy tasks of relay assignment and storage of information about the network. Using privileged nodes, k2pruning could become a viable choice since a lot of the data about neighbourhoods could be offloaded to such a node.

This would fundamentally change the outcome of the project since the k2pruning showed to outperform the HighConnect in Section 6.5.

Further the HighConnect could offload some of its process intensive tasks to a privileged node and on sparse networks a smaller amount of relays could be made.

However this also gives some issues of edge cases in the network setup. When information is offloaded to privileged nodes then not all information about the network is relayed and the privileged nodes would have to communicate internally with each other to make sure the whole network gets connected properly. This could be solved in multiple different ways either by having another technology to communicate on such as Wi-Fi or it could be that nodes had some way of connecting edge cases themselves. Throughout the project, countermeasures have been taken to avoid getting outdated information. If the nodes used reliable messages it could guarantee that nodes got the newest information. This could be done by first finding neighbours as before. Then when distributing information and coordinating between each others, reliable messages could be used to ensure that one node gets all information about the network it needs. This will potentially result in a better network, with the cost that it would take more time than with unreliable messages.

Conclusion 10

In recent years the electronic industry has worked towards building the Internet of Things (IoT). IoT is a industry term for connecting electrical devices and appliances to each other and to the internet. For this the Bluetooth Special Interest Group (BSIG) have made a new standard called Bluetooth Mesh (BM). This project has received an early build of the standard from Samsung to try and build a self configuring network.

In Chapter 2 the project starts out with analysing the BM protocol stack. It was chosen to look at the lower layers of the BM protocol stack namely the Physical Layer (PL), Link Layer (LL), and Network Layer (NL). From the acquired knowledge a Bluetooth Mesh use case analysis was conducted. The analysis showed that the BM network should keep relay quantity low, see Section 2.6.2.

Based on this discovery a problem formulation and a delimitation was made, see Chapter 3. The project decided to focus on different algorithms which tried to keep quantity of relays down through creating a minimised Connected Dominating Sets (CDSs). In Chapter 4 the advantages and disadvantages of using central and distributed algorithms were discussed. The project chose to focus primarily on distributed solutions.

For comparing the strengths and weaknesses of different relay distribution algorithms, a list of performance metrics was made, see Section 4.3.

For comparing the different algorithms a simulator was developed because no other test methods were feasible. The simulator was written in MATLAB which was arbitrarily chosen. The simulator was made using the IEEE 802.15.4a channel model for path loss and the BM standard. This was done in Chapter 5. For validating that the simulator was correct, three scenarios were simulated and tested, see Section 5.6. Here the simulator was deemed correct and different relay distribution algorithms could be tested.

In Chapter 6 different algorithm were described and compared to each other. The algorithm were compared by looking at how their resulting networks would perform. Because of this, it was desirable to avoid implementation specific variations. Nodes did therefore always have updated and complete information about their neighbourhood or the whole network, depending on the used algorithm. The different algorithms were compared by estimating the quantity transmissions needed to setup the network, their storage needs, and their Packet Delivery Ratios (PDRs). All the algorithms showed to outperform the all nodes are relays baseline. The K2Pruning algorithm was shown to outperform all other algorithms. The K2Pruning algorithm has high requirements in terms of node storage. The second best performing algorithm, highest connectivity based greedy algorithm (HighConnect), was therefore chosen as a compromise between node complexity and network performance.

Because the implementation specific variations were not considered in Chapter 6, the HighConnect algorithm was implemented on the nodes. This was done to test if the algorithm could perform as well, when nodes had to gather information through packet transmissions. Chapter 7 starts with an explanation of the problems the algorithm faces. The problems were discovery of neighbourhood, avoiding simultaneous conflicting decisions, synchronising setup phases, and lowering risk of making decisions based on incomplete or outdated information.

Due to time constraints, the algorithm was only designed with a 200 node 60 m by 60 m network in mind. The results in the report are therefore only valid for such a network.

The problems were then solved by introducing two stages, one for discovery and one for relay distribution, where simultaneous decisions are avoided and steps kept synchronised. The probability of making decisions based on outdated or incomplete information was lowered in the relay distribution phase by adding a delay based on measured network parameters.

For the discovery of neighbours (Section 7.4), three different approaches were discussed: reading Received Signal Strength Indicator (RSSI) and if it is within a threshold then nodes are neighbours, receiving a setup packet then they are neighbours, nodes are neighbours if the PDR between two nodes is above a threshold. Through analysing the three methods it was found that measuring PDRs and comparing them to a threshold provided good properties in return for longer setup times. This compromise was deemed acceptable. The PDR is measured by injecting at least 40 packets into the network. The quantity of received packets is used to determine the PDR. This also means that the discovery stage takes 161 s before the 200 nodes have found their neighbours.

When the nodes have found their neighbours, the distribution of relays begins. The relay distribution stage follows the algorithms overall structure however it was important to make decisions happen individually so that no conflicting decisions happened simultaneously.

The solution for this was to make nodes wait on each other by introducing a wait list. The wait list would contain the ID's of all nodes who had marked other nodes to be relays. Through this, the implementation made sure no two neighbouring nodes would make conflicting decisions. The wait list also did so that if a node had a neighbour on a lower step, it would wait until that neighbour had caught up with the rest of the neighbourhood (Section 7.5).

The implementation was simulated in Chapter 8. The simulation showed that the implemented algorithm had on average 2 relays more than the theoretical case from Chapter 6. The implementation's PDR results showed that it was statistically equal to the theoretical. This shows that the implemented algorithm performs similar to the theoretical even though it has more relays. It was shown that the algorithm could connect a 200 node network by distributing relays. It is determined that this project was a success because of how equal the implementation was to the theoretical.

Further study should design algorithm to work on different network densities and could include aspects such as network maintenance after setup.

Bibliography

[1] Bluetooth SIG Group. *Bluetooth Core Specification v 5.0*, 2017. URL https://www.bluetooth.com/specifications/adopted-specifications. (Cited on pages v, 6, 7, 8, 9, 10, 11, 12, 13, 15, and 75.)

[2] Bluetooth SIG Group. *Bluetooth Core Mesh Specification*, 2017 - d09r19. (Cited on pages v, 4, 5, 6, 14, 15, 16, 20, and 43.)

[3] Jacob Kastrenakes. This is samsung's grand vision for the internet of things. Website, 2015. URL https://www.theverge.com/2015/1/5/7497537/samsung-iot-internet-of-things-vision-presented-at-ces-2015-keynote. (Cited on page 1.)

[4] European Communication Office. Eco frequency information system. Website, 2017. URL http://efis.dk/. (Cited on page 1.)

[5] Radio Electronics. Wi-fi / wlan channels, frequencies, bands and bandwidths. Website, 2016. URL http://www.radio-electronics.com/info/wireless/wi-fi/80211-channels-number-frequencies-bandwidth.php. (Cited on page 7.)

[6] ETSI. *ETSI EN 300 328 V1.9.1 (2015-02)*, 2015. URL http://www.etsi.org/deliver/etsi_en/300300_300399/300328/01.09.01_60/en_300328v010901p.pdf. (Cited on page 7.)

[7] Simon Haykin. *Communication Systems*. John Wiley & Sons, Inc., 4 edition, 2001. ISBN 0-471-17869-1. (Cited on page 8.)

[8] Kazuaki Murota and Kenkichi Hirade. Gmsk modulation for digital mobile radio telephony. *IEEE Transactions on Wireless Communications*, 29(7):1044–1050, 1981. doi: 0090-6778/81/0700-1044$00.75. URL http://www.bertozi.com/adb/Radioamadorismo/GMSK%20Modulation%20for%20Digital%20Mobile%20Radio%20Telephony.pdf. (Cited on pages 10, 39, and 40.)

[9] Rajendra K. Jain, Dah-Ming W. Chiu, and William R. Hawe. A quantitative measure of fairness and discrimination for resource allocation in shared computer system. Technical report, 1984. URL http://www.cse.wustl.edu/~jain/papers/ftp/fairness.pdf. (Cited on page 32.)

[10] Andreas F. Molisch, Kannan Balakrishnan, Dajana Cassioli, Chia-Chin Chong, Shahriar Emami, Andrew Fort, Johan Karedal, Juergen Kunisch, Hans Schantz, Ulrich Schuster, and Kai Siwiak. Ieee 802.15.4a channel model - final report. Technical report, 2005. URL http://www.ieee802.org/15/pub/04/15-04-0662-01-004a-channel-model-final-report-r1.pdf. (Cited on pages 35 and 36.)

[11] AT & T. Antenna fundamentals - technical brief. Technical report, 2009. URL https://www.att.com/edo/en_US/pdf/AntennaFundamentals.pdf. (Cited on page 36.)

[12] A. Iyer, C. Rosenberg, and A. Karnik. What is the right model for wireless channel interference? *IEEE Transactions on Wireless Communications*, 8(5):2662–2671, 2009. doi: 10.1109/TWC.2009.080720. URL http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=4927481. (Cited on page 38.)

[13] H. Lim and C.Kim. Flooding in wireless ad hoc networks. *Computer Communications*, 24:353–363, 2001. URL http://ac.els-cdn.com/S0140366400002334/1-s2.0-S0140366400002334-main.pdf?_tid=f61dad9a-2d88-11e7-869f-00000aacb35d&acdnat=1493545372_ebb1c2330f9c206ee23befbcf4b9915a. (Cited on page 50.)

[14] S. Guha and Khuller. Approximation algorithms for connected dominating sets. *Algorithmica*, 20:374–387, 1998. doi: 10.1007/PL00009201. URL http://link.springer.com/article/10.1007/PL00009201. (Cited on page 50.)

[15] Said Ghnimi, Adnen Rajhi, and Ali Gharsallah. A distributed greedy algorithm for constructing connected dominating sets in wireless sensor networks. Technical report, N/A. URL http://webpages.ursinus.edu/nscoville/sensornets14.pdf. (Cited on page 52.)

[16] Jie Wu and Hailan Li. On calculating connected dominatin set for efficient routing in ad hoc wireless networks. Technical report, 1999. URL http://www.distcomp.ethz.ch/alumni/pascal/refs/ds_1999_wu.pdf. (Cited on page 55.)

[17] R. Wattenhofer. "dominator algorithm". Website, 2004. URL http://www.dcg.ethz.ch/lectures/ss04/mobicomp/lecture/8/Chapter8DominatingSets4Slides.pdf. (Cited on page 57.)

[18] Petar Popovski. statistics mm4: hypothesis testing, part i. Moodle, 2017. URL https://www.moodle.aau.dk/pluginfile.php/883483/course/section/295297/slides_mm4_2017.pdf. (Cited on page 74.)

[19] Sheldon M. Ross. *Introduction to Probability and Statistics For Engineers and Scientists*. ELSEVIER, 5 edition, 2014. ISBN 978-0-12-394811-3. (Cited on page 81.)

[20] Douglas C. Montgomery and George C Runger. *Applied statistics and probability for engineers*. John Wiley and Sons, 1 edition, 1994. ISBN 0-471-54041-2. (Cited on page 91.)

Approximation of traffic intensity

A

This appendix will approximate the number of transmissions coming from the box network from Section 2.6.2, dependent on the boxes injection frequency f_I and number of relays nr. Due to symmetry the traffic intensity from boxes 1 and 4 are equal, just like how boxes 2 and 3 are equal. The probability of successfully injecting a packet into the network, dependent on the traffic intensity ti is denoted $q_i(ti)$. The probability of successfully relaying a packet into the network, dependent on the quantity of relays nr and the traffic intensity ti is denoted $q_r(ti, nr)$.

Traffic intensity from box 1

The box 1 does not contain any relays, and will therefore only inject packets. A box injects f_I packets per second. The traffic intensity from box 1 is therefore given by Equation (A.1).

$$ti_1 = f_I \tag{A.1}$$

Traffic intensity from box 2

Nodes in box 2 will transmit their own injections, as well as packets received by the relays. The probability of a packet reaching the relays in box 2 is different dependent on the origin of the packet. The traffic from the different boxes reaching the relays in box 2 is now approximated.

Box 1

For a packet to reach the relays in box 2 from box 1, it must be injected from box 1 to 2, as illustrated on Figure A.1.

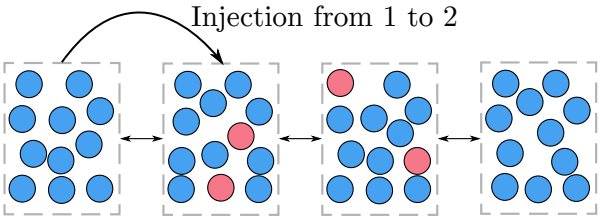


Figure A.1: Illustration showing paths from box 1 to box 2

The traffic received by relay nodes in box 2 from nodes in box 1 is given by Equation (A.2).

$$\text{trafficFrom}(1) = q_i(ti_1 + ti_2 + ti_3) \cdot f_I \tag{A.2}$$

Box 2

For a packet to reach the relays in box 2 from box 2, it must be either be injected from box 2 to box 2, or injected from box 2 to box 3 and relayed from box 3 to box 2, as illustrated on Figure A.2.

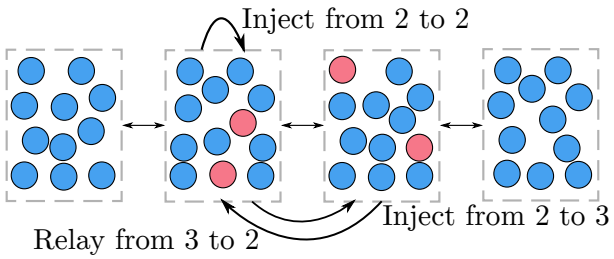


Figure A.2: Illustration showing paths from box 2 to box 2

The traffic received by relay nodes in box 2 from nodes in box 2 is given by Equation (A.3).

$$\text{trafficFrom}(2) = \big(q_i(ti_1 + ti_2 + ti_3) + q_i(ti_2 + ti_3 + ti_4) \cdot (1 - q_i(ti_1)) \cdot q_r(ti_1 + ti_2 + ti_3)\big) \cdot f_I \tag{A.3}$$

Box 3

For a packet to reach the relays in box 3 from box 2, it must be either be injected from box 3 to box 2, or injected from box 3 to box 3 and relayed from box 3 to box 2, as illustrated on Figure A.3.

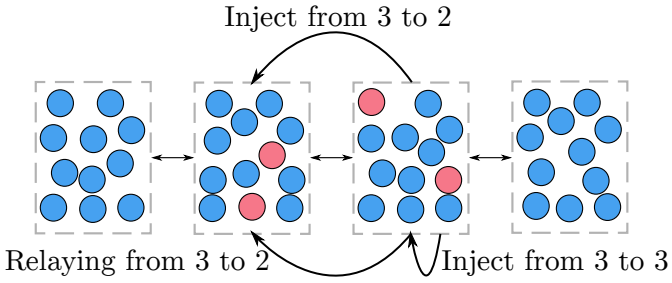


Figure A.3: Illustration showing paths from box 3 to box 2

The traffic received by relay nodes in box 2 from nodes in box 3 is given by Equation (A.4).

$$\text{trafficFrom}(3) = \big(q_i(ti_1 + ti_2 + ti_3) + q_i(ti_2 + ti_3 + ti_4) \cdot (1 - q_i(ti_1)) \cdot q_r(ti_1 + ti_2 + ti_3)\big) \cdot f_I \tag{A.4}$$

Box 4

For a packet to reach the relays in box 2 from box 4, it must be injected from box 4 to box 2 and relayed from box 3 to box 2, as illustrated on Figure A.4.

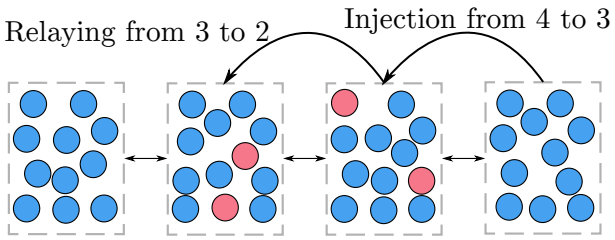


Figure A.4: Illustration showing paths from box 4 to box 2

The traffic received by relay nodes in box 2 from nodes in box 4 is given by Equation (A.5).

$$\text{trafficFrom}(4) = \big(q_i(ti_2 + ti_3 + ti_4) \cdot q_r(ti_1 + ti_2 + ti_3)\big) \cdot f_I \tag{A.5}$$

Total transmissions from box 2

The total quantity of transmissions from box 2 is then given by Equation (A.6).

$$\begin{aligned} ti_2 &= \left(\sum_{n=1}^{4} \text{trafficFrom}(n)\right) \cdot nr + \text{ownInjections} \\ &= \left(\sum_{n=1}^{4} \text{trafficFrom}(n)\right) \cdot nr + f_I \end{aligned} \tag{A.6}$$

By defining $ti_s = ti_1 + ti_2 + ti_3$, which due to symmetry is equal to $ti_2 + ti_3 + ti_4 = ti_s = ti_1 + ti_2 + ti_3$, and $q_{mp}(ti_1, ti_s, nr) = q_i(ti_s) + q_i(ti_s) \cdot (1 - q_i(ti1)) \cdot q_r(ti_s, nr)$, ti_2 can be reduced to Equation (A.7)

$$ti2 = f_I \cdot \big(q_i(ti_s)\, nr + q_i(ti_s)\, q_r(ti_s) + 2\, nr\, q_{mp}(ti_1, ti_s, nr) + 1\big) \qquad [\text{s}^{-1}] \tag{A.7}$$

Packet arrival probability for simulator test 2

B

This appendix approximates the probability of successfully transmitting a packet from node 1 to the relay node 2, in the test conducted in Section 5.6.2

When node 1 transmits to the relay node 2, there are nine different outcomes. The relay node can be on any of the three channels before receiving a packet, and the relay can choose any of the three channels after relaying a packet. This adds up to 3^2 different outcomes.

The probability that the relay receives a transmission from node 1, is given by Equation (B.1)

$$q = Pr(\text{Relay receives}|\text{Relay is listening}) \cdot \left(1 - q \cdot \left(\frac{1}{3}q_1 + \frac{1}{3}q_2 + \frac{1}{3}q_3\right)\right)[1] \quad \text{(B.1)}$$

$$q_1 = \left(\frac{1}{3}q_{11} + \frac{1}{3}q_{12} + \frac{1}{3}q_{13}\right)$$
$$q_2 = \left(\frac{1}{3}q_{21} + \frac{1}{3}q_{22} + \frac{1}{3}q_{23}\right)$$
$$q_3 = \left(\frac{1}{3}q_{31} + \frac{1}{3}q_{32} + \frac{1}{3}q_{33}\right)$$

Where:

q	is the probability of successfully transmitting a packet from node 1 to node 2	[1]
q_{mn}	is the probability that the relay is listening when it is listening on channel m before relaying and n after relaying.	[1]

The probabilities q_{mn} are derived by analysing the nine different cases. However before the cases are presented, a constant I is introduced by Equation (B.2).

$$I = \left(\left(\frac{2 \cdot k}{n} - (47 \cdot 3 + 4)\right) - (47 \cdot 3 + 4)\right) = \frac{2 \cdot k}{n} - 290 \quad \text{(B.2)}$$

Where:

I	is the length of the interval where the next packet is scheduled, see Section 5.6.2	[tick]
k	is the quantity of simulator tick in a second	[tick /s]
n	is the packet injections per second	[packet /s]

q_{11}

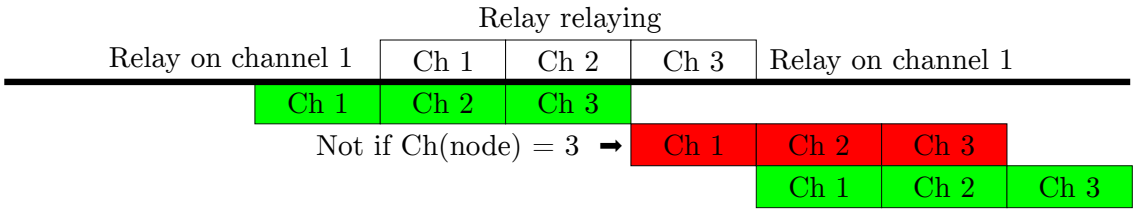


Figure B.1: Illustration of possible outcomes if the relay is listening on channel 1 before packet receipt and channel 1 after relaying. Red boxes denotes lost packets, green boxes denotes successful transmissions.

Figure B.1 illustrates what happens if the relay is listening on channel 1, receives at packet, transmits it, and returns to listen on channel 1. If node 1 begins transmission of the next packet, within the time frame where the relay is transmitting on channel 1, the packet it lost. If the non receipt breaking node behaviour, node 1 can only begin transmission within this time frame if some conditions are met: node 1 returns to channel 1 or 2 after transmitting or node 1 begins transmissions of the next packet on the following simulator tick after finishing its last transmission. Equation (B.3) states this probability.

$$q_{11,\text{non receipt breaking}} = \frac{2}{3} \cdot \frac{tx}{I} + \frac{1}{3} \cdot \frac{1}{I} \quad [1] \quad \text{(B.3)}$$

Where:

tx	is the length of a packet in simulator ticks	[tick]

In the case where node are allowed to break receipt, node 1 can always begin transmission within the duration where the relay is transmitting on channel 3, thus causing packet loss. Equation (B.4) states this probability

$$q_{11,\text{receipt breaking}} = \frac{3}{3} \cdot \frac{tx}{I} \quad [1] \quad \text{(B.4)}$$

q_{12}

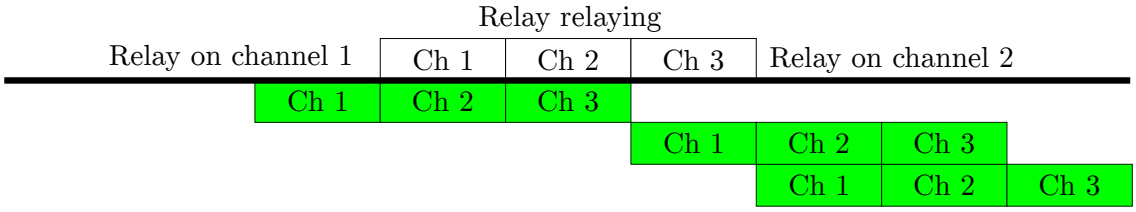


Figure B.2: Illustration of possible outcomes if the relay is listening on channel 1 before packet receipt and channel 2 after relaying. Red boxes denotes lost packets, green boxes denotes successful transmissions.

Using Figure B.2, Equations (B.5) and (B.6) are derived.

$$q_{12,\text{non receipt breaking}} = \frac{3}{3} \cdot 0 \quad [1] \quad \text{(B.5)}$$

$$q_{12,\text{receipt breaking}} = \frac{3}{3} \cdot 0 \quad [1] \quad \text{(B.6)}$$

q_{13}

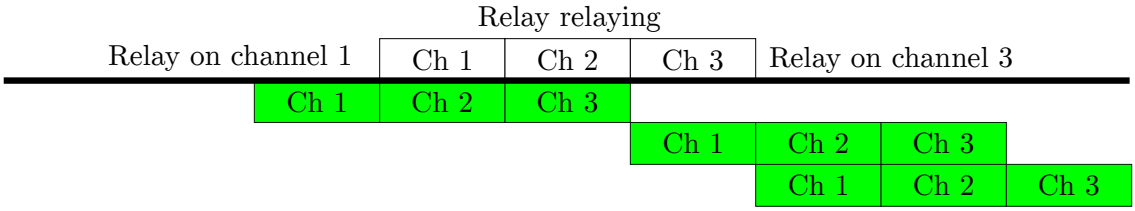


Figure B.3: Illustration of possible outcomes if the relay is listening on channel 1 before packet receipt and channel 3 after relaying. Red boxes denotes lost packets, green boxes denotes successful transmissions.

Using Figure B.3, Equations (B.7) and (B.8) are derived.

$$q_{13,\text{non receipt breaking}} = \frac{3}{3} \cdot 0 \quad [1] \quad \text{(B.7)}$$

$$q_{13,\text{receipt breaking}} = \frac{3}{3} \cdot 0 \quad [1] \quad \text{(B.8)}$$

q_{21}

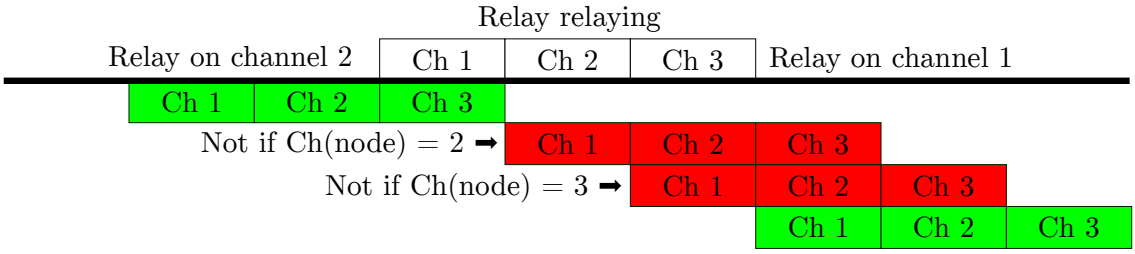


Figure B.4: Illustration of possible outcomes if the relay is listening on channel 2 before packet receipt and channel 1 after relaying. Red boxes denotes lost packets, green boxes denotes successful transmissions.

Using Figure B.4, Equations (B.9) and (B.10) are derived.

$$q_{21,\text{non receipt breaking}} = \frac{1}{3} \cdot \frac{2 \cdot tx}{I} + \frac{2}{3} \cdot \frac{tx+1}{I} \quad [1] \quad \text{(B.9)}$$

$$q_{21,\text{receipt breaking}} = \frac{3}{3} \cdot \frac{2 \cdot tx}{I} \quad [1] \quad \text{(B.10)}$$

q_{22}

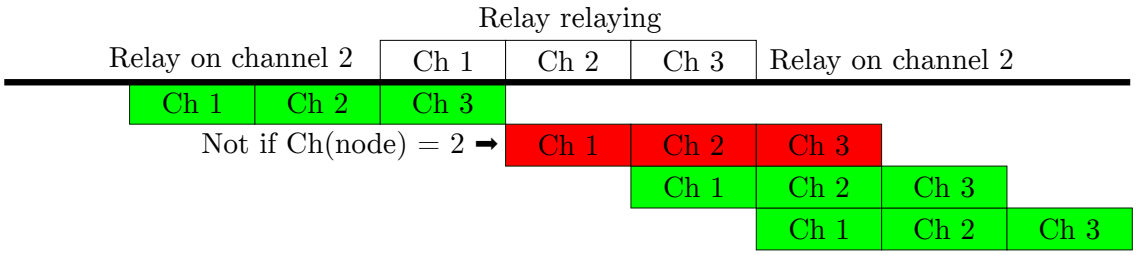


Figure B.5: Illustration of possible outcomes if the relay is listening on channel 2 before packet receipt and channel 2 after relaying. Red boxes denotes lost packets, green boxes denotes successful transmissions.

Using Figure B.5, Equations (B.11) and (B.12) are derived.

$$q_{22,\text{non receipt breaking}} = \frac{2}{3} \cdot \frac{tx}{I} + \frac{1}{3} \cdot \frac{1}{I} \quad [1] \quad \text{(B.11)}$$

$$q_{22,\text{receipt breaking}} = \frac{3}{3} \cdot \frac{tx}{I} \quad [1] \quad \text{(B.12)}$$

q_{23}

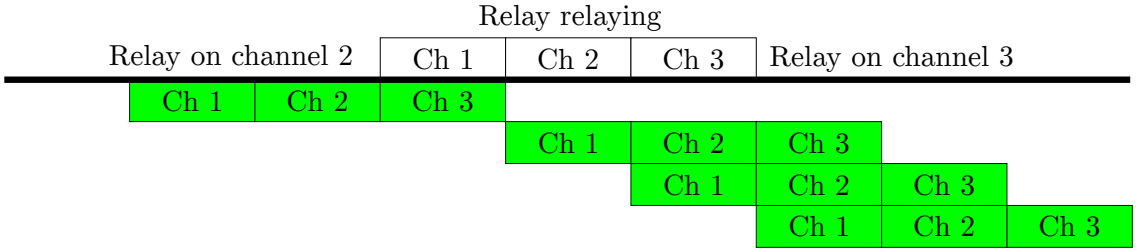


Figure B.6: Illustration of possible outcomes if the relay is listening on channel 2 before packet receipt and channel 3 after relaying. Red boxes denotes lost packets, green boxes denotes successful transmissions.

Using Figure B.6, Equations (B.13) and (B.14) are derived.

$$q_{23,\text{non receipt breaking}} = \frac{3}{3} \cdot 0 \quad [1] \quad \text{(B.13)}$$

$$q_{23,\text{receipt breaking}} = \frac{3}{3} \cdot 0 \quad [1] \quad \text{(B.14)}$$

q_{31}

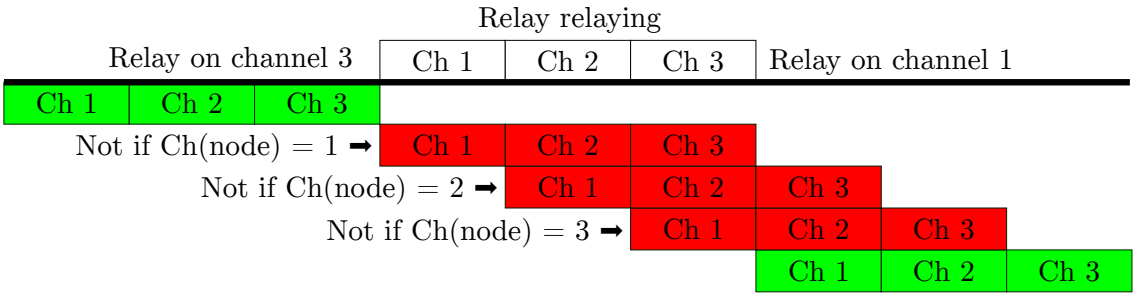


Figure B.7: Illustration of possible outcomes if the relay is listening on channel 3 before packet receipt and channel 1 after relaying. Red boxes denotes lost packets, green boxes denotes successful transmissions.

Using Figure B.7, Equations (B.15) and (B.16) are derived.

$$q_{31,\text{non receipt breaking}} = \frac{3}{3} \cdot \frac{2 \cdot tx + 1}{I} \quad [1] \quad \text{(B.15)}$$

$$q_{31,\text{receipt breaking}} = \frac{3}{3} \cdot \frac{3 \cdot tx}{I} \quad [1] \quad \text{(B.16)}$$

q_{32}

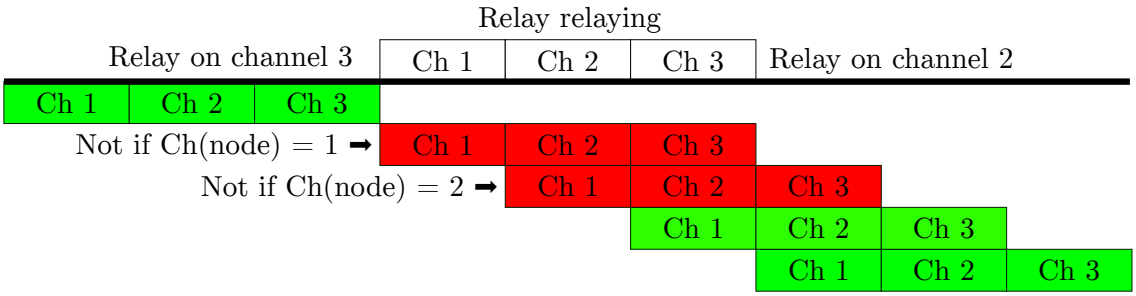


Figure B.8: Illustration of possible outcomes if the relay is listening on channel 3 before packet receipt and channel 2 after relaying. Red boxes denotes lost packets, green boxes denotes successful transmissions.

Using Figure B.8, Equations (B.17) and (B.18) are derived.

$$q_{32,\text{non receipt breaking}} = \frac{2}{3} \cdot \frac{tx + 1}{I} + \frac{1}{3} \cdot \frac{2 \cdot tx}{I} \quad [1] \quad \text{(B.17)}$$

$$q_{32,\text{receipt breaking}} = \frac{3}{3} \cdot \frac{2 \cdot tx}{I} \quad [1] \quad \text{(B.18)}$$

q_{33}

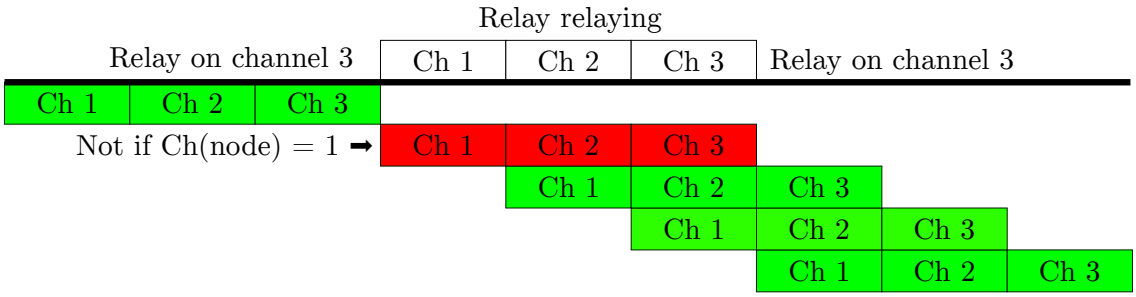


Figure B.9: Illustration of possible outcomes if the relay is listening on channel 3 before packet receipt and channel 3 after relaying. Red boxes denotes lost packets, green boxes denotes successful transmissions.

Using Figure B.9, Equations (B.19) and (B.20) are derived.

$$q_{33,\text{non receipt breaking}} = \frac{1}{3} \cdot \frac{1}{I} + \frac{2}{3} \cdot \frac{tx}{I} \quad [1] \quad \text{(B.19)}$$

$$q_{33,\text{receipt breaking}} = \frac{3}{3} \cdot \frac{tx}{I} \quad [1] \quad \text{(B.20)}$$

Collecting the equations

Collecting the equations gives Equations (B.21) and (B.22).

$$q_{\text{non receipt breaking}} = \frac{1.8 \cdot k - n \cdot \left(261 + (2 \cdot tx + 1) \cdot \frac{q_{\text{non receipt breaking}}}{3}\right)}{2 \cdot k - 290 \cdot n} \quad [1] \quad \text{(B.21)}$$

$$q_{\text{receipt breaking}} = \frac{1.8 \cdot k - n \cdot \left(261 + tx \cdot q_{\text{receipt breaking}}\right)}{2 \cdot k - 290 \cdot n} \quad [1] \quad \text{(B.22)}$$

Equation (B.21) does not consider the probability of another outcome where packet loss can also happen. If Node 1 has just successfully transmitted a packet to node 2, node 2 will begin relaying immediately. If node 1 receives the first byte of node 2's relaying, it can not break receipt. If node 1 has scheduled its next transmission within the duration of node 2's relaying, it will begin its own transmission right after finishing receipt. This will increase the probability of packet loss, as the size of I decreases. Equations (B.21) and (B.22) therefore serves as upper and lower bounds of $Pr(N2_{Rx}|N1_{Tx})$, as stated in Equation (B.23).

$$q_{\text{non receipt breaking}} \geq Pr(N2_{Rx}|N1_{Tx}) \geq q_{\text{receipt breaking}} \quad \text{(B.23)}$$

Where:

$Pr(N2_{Rx}|N1_{Tx})$ is the probability that node 2 successfully receives given that node 1 is transmitting [1]

Simulation parameters C

Table C.1: Parameter options that define the behaviour of the simulator.

Name	**Purpose**	**Range of value / Default value**
amountNodes	Amount of nodes that are simulated	Any positive integer / 1000
lenX	Length of the x-axis in meters	Any positive float / 60
lenY	Length of the y-axis in meters	Any positive float / 60
staticTest	Makes the simulator static. Assumes that there are already relays, and generates messages which the nodes then send to each other. Is faster than if the nodes dynamically need to send messages	Boolean / 0
rngDefault	Sets the random number generator in MATLAB to 'default', giving reproducable results. Mainly used for developing and debugging.	Boolean / 0
gpu	Enables using a graphics card for accelerating simulation of the physical layer.	Boolean / 0
xPL	Scales the power output of all nodes. An increased value increases the pathloss, and thus decreases the range	Any integer / 15
LOSLimit	Sets the limit in terms of Block Error Probability (BLEP) for which the ideal algorithms will detect 2 nodes being able to transmit to each other.	$]0, 1[$ float / 0.1
trafficLevel	Sets the amount of messages that are sent per second	Any positive integer / 1000
messageLength	Sets length of messages	31-47 integer / 47
timeInSeconds	Sets time that the simulator runs in seconds	Any positive value / 1

numMessages	Number of messages that will be sent, calculated by numMessages = $\lceil \text{trafficLevel} \cdot \text{timeInSeconds} \rceil$	Any positive integer / 1000
globalTTL	Sets the TTL that the node uses as default	2-127 integer / 127
msgQEnabled	Enables or disables the message queue	Boolean / 0
msgQMax	Sets the maximum messages that may be in the message queue	Any positive integer / 100
minChannelChangeWai	When a node is done transmitting on a channel, the node can wait a random amount of time before transmitting on the next channel. This value is the minimum amount of ticks that it can take.	Any positive integer / 10
maxChannelChangeWait	See above, this value is the maximum amount of ticks that a node can wait before transmitting on a channel.	Any positive integer / 100
minWaitAfterReceive	When a node has received a message and decided to relay, it waits a random amount of ticks, this is the minimum amount of time.	Any positive integer / 10
maxWaitAfterReceive	When a node has received a message and decided to relay, it waits a random amount of ticks, this is the maximum amount of time.	Any positive integer / 200
reuseNodefile	To run a simulation with the same nodes as are already present, set this to 1.	Boolean / 0
reusepathLossModel	To run a simulation with the same pathloss as are already present, set this to 1.	Boolean / 0
receiverSensitivity	Effectively sets the minimum power in which a node can receive data	Any float / -70
PathLossModel	The type of model that should be used to model the pathloss. Can be freespace, office or residential	String / 'office'
PL_NLOS_mean	Defines the mean, when deciding on NLOS or LOS, see Section 5.2.1	Any positive float / 10
PL_NLOS_variance	Defines the variance, when deciding on NLOS or LOS, see Section 5.2.1	Any positive float / 5
relayAlg	Specifies which algorithm is used to pick relays in 'ideal' case, see Chapter 6	String / 'none'

AGIF	If 1, saves information about the simulator in order to later plot.	Boolean / 0
relayAlgPath	Path to relay information if relays are already chosen	String
endMsg	Specifies how many messages a node should have sent before stopping the simulator.	Positive integer
fairness	If 1, then extra messages are created from nodes that are furthest from each other.	Boolean / 0
saveEnabled	If 1, then saving is done	Boolean / 1
saveFrequency	How many ticks there should be between saves	Positive integer /100000
savePath	Path to save file	string / `"/saves_/save_"`
illustrationOnly	Set to 1 if only illustration is needed.	Boolean /0
messageCacheLength	Amount of messages that the cache has space for	Positive integer / 100

Relay distributions for a 330 m by 330 m network

D

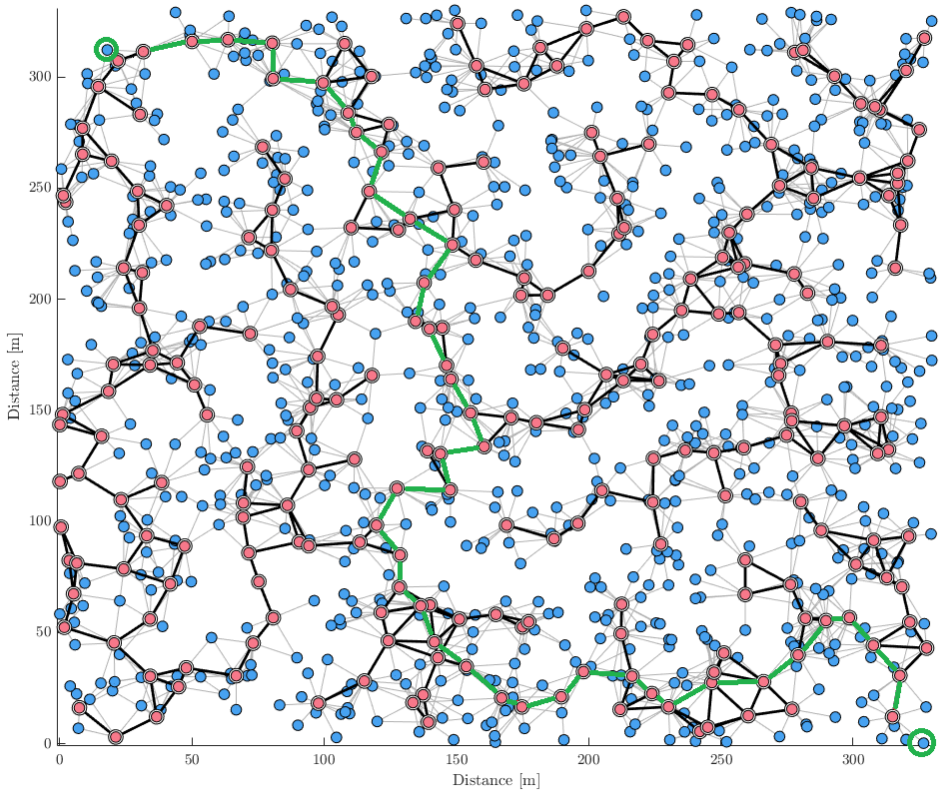


Figure D.1: Node distributions for a 330 m by 330 m network after running the HighConnect algorithm.

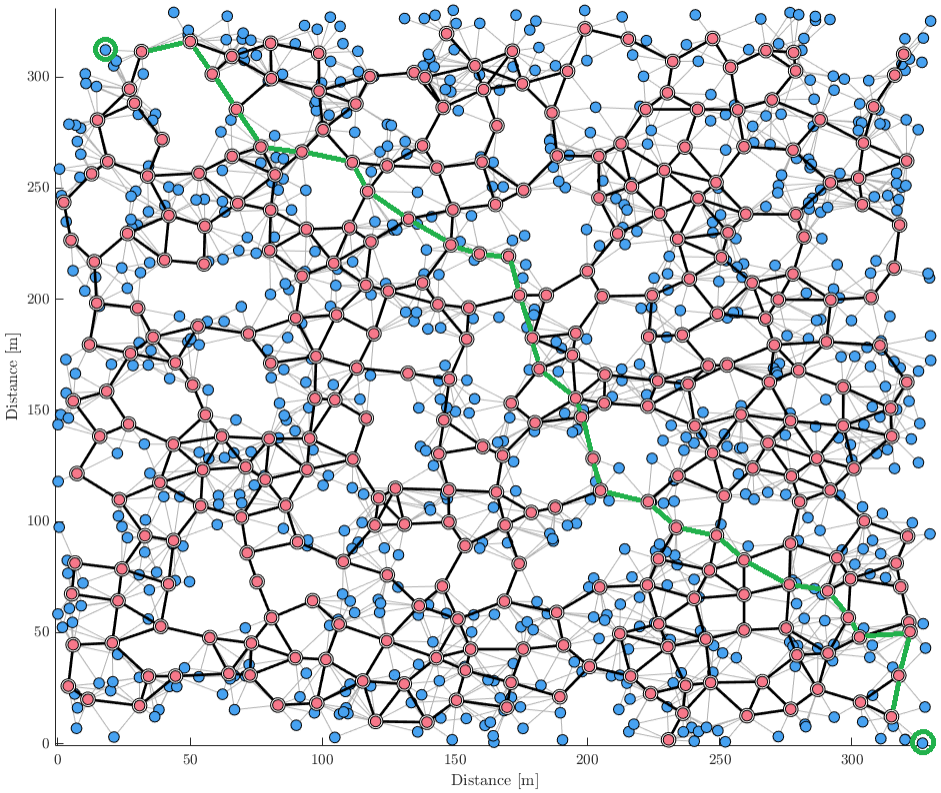


Figure D.2: Node distributions for a 330 m by 330 m network after running the K2Pruning algorithm.

Simulation journal: Evaluation of different algorithms

E

Date: 05/05-2017

Purpose

The purpose of this simulation is to test the viability of different algorithms. This is done by simulating each algorithm at multiple traffic levels. For each traffic level, 4 simulations are done. One set of simulations are done with the `dense` network, and another set of simulations are done with the `sparse` network. The `dense` network uses an area of 60 m by 60 m and the `sparse` network uses 150 m by 150 m.

Simulation parameters

The simulation parameters used for simulation are given in Table E.1, Table E.2, and Table E.3. **Nodegen** specifies which seed[1] to run during the simulation. **Traffic** specifies which seed is used to generate traffic, such as timings between nodes and who send to who. **Topology** specifies which seed is used for the relay algorithms. **Fairness** specifies if there needs to be sent extra packets from the nodes furthest away from each other. **Dense** specifies if a 60 m by 60 m area or a 150 m by 150 m are is used (1 for 60 m by 60 m, 0 for 150 m by 150 m). **TLIndex** is used to specify which traffic level intensities are simulated. **SimIndex** is used to keep track on which simulation is which, also specifies where a savefile will be. **TL** is which traffic level intensities that are simulated. **Script** is the name of the script that should be run in order to simulate.

In Table E.3, **Length** specifies what the length of one of the sides of the quadratic are are. So a length of 60 specifies 60 m by 60 m area.

[1] Here, a seed specifies which number is used to seed the Mersenne Twister pseudo random number generator.

Table E.1: PDR simulation parameters

Nodegen	Traffic	Topology	Fairness	Dense	TLIndex	SimIndex	TL	Script
200	200	200	0	1	1	61	5 7 10 14	st61.sh
200	200	200	0	1	5	62	19 27 37 52	st62.sh
200	200	200	0	1	9	63	73 102 143 200	st63.sh
200	200	200	0	0	1	64	5 7 10 14	st64.sh
200	200	200	0	0	5	65	19 27 37 52	st65.sh
200	200	200	0	0	9	66	102 143 200	st66.sh
220	200	200	0	1	1	67	5 7 10 14	st67.sh
220	200	200	0	1	5	68	19 27 37 52	st68.sh
220	200	200	0	1	9	69	73 102 143 200	st69.sh
220	200	200	0	0	1	70	5 7 10 14	st70.sh
220	200	200	0	0	5	71	19 27 37 52	st71.sh
220	200	200	0	0	9	72	102 143 200	st72.sh
240	200	200	0	1	1	73	5 7 10 14	st73.sh
240	200	200	0	1	5	74	19 27 37 52	st74.sh
240	200	200	0	1	9	75	73 102 143 200	st75.sh
240	200	200	0	0	1	76	5 7 10 14	st76.sh
240	200	200	0	0	5	77	19 27 37 52	st77.sh
240	200	200	0	0	9	78	102 143 200	st78.sh

Table E.2: Worst case PDR simulation parameters

Nodegen	Traffic	Topology	Fairness	Dense	TLIndex	SimIndex	TL	Script
200	200	200	1	1	1	101	5 7 10 14	st101.sh
200	200	200	1	1	5	102	19 27 37 52	st102.sh
200	200	200	1	1	9	103	73 102 143 200	st103.sh
200	200	200	1	0	1	104	5 7 10 14	st104.sh
200	200	200	1	0	5	105	19 27 37 52	st105.sh
200	200	200	1	0	9	106	102 143 200	st106.sh
220	200	200	1	1	1	107	5 7 10 14	st107.sh
220	200	200	1	1	5	108	19 27 37 52	st108.sh
220	200	200	1	1	9	109	73 102 143 200	st109.sh
220	200	200	1	0	1	110	5 7 10 14	st110.sh
220	200	200	1	0	5	111	19 27 37 52	st111.sh
220	200	200	1	0	9	112	102 143 200	st112.sh
240	200	200	1	1	1	113	5 7 10 14	st113.sh
240	200	200	1	1	5	114	19 27 37 52	st114.sh
240	200	200	1	1	9	115	73 102 143 200	st115.sh
240	200	200	1	0	1	116	5 7 10 14	st116.sh
240	200	200	1	0	5	117	19 27 37 52	st117.sh
240	200	200	1	0	9	118	102 143 200	st118.sh

Table E.3: Comparison between K2Pruning and HighConnect simulation parameters

Nodegen	Traffic	Topology	Fairness	Length	SimIndex	script
200	200	200	1	60	11	st11.sh
230	200	200	1	60	12	st12.sh
260	200	200	1	60	13	st13.sh
290	200	200	1	60	14	st14.sh
320	200	200	1	60	15	st15.sh
360	200	200	1	60	16	st16.sh
380	200	200	1	60	17	st17.sh
410	200	200	1	60	18	st18.sh
460	200	200	1	60	19	st19.sh
470	200	200	1	60	20	st20.sh
500	200	200	1	60	21	st21.sh
540	200	200	1	60	22	st22.sh
200	200	200	1	150	23	st23.sh
230	200	200	1	150	24	st24.sh
260	200	200	1	150	25	st25.sh
290	200	200	1	150	26	st26.sh
320	200	200	1	150	27	st27.sh
360	200	200	1	150	28	st28.sh
380	200	200	1	150	29	st29.sh
410	200	200	1	150	30	st30.sh
460	200	200	1	150	31	st31.sh
470	200	200	1	150	32	st32.sh
500	200	200	1	150	33	st33.sh
540	200	200	1	150	34	st34.sh
200	200	200	1	240	35	st35.sh
230	200	200	1	240	36	st36.sh
260	200	200	1	240	37	st37.sh
290	200	200	1	240	38	st38.sh
320	200	200	1	240	39	st39.sh
360	200	200	1	240	40	st40.sh
380	200	200	1	240	41	st41.sh
410	200	200	1	240	42	st42.sh
460	200	200	1	240	43	st43.sh
470	200	200	1	240	44	st44.sh
500	200	200	1	240	45	st45.sh
540	200	200	1	240	46	st46.sh
200	200	200	1	330	47	st47.sh
230	200	200	1	330	48	st48.sh
260	200	200	1	330	49	st49.sh
290	200	200	1	330	50	st50.sh
320	200	200	1	330	51	st51.sh
360	200	200	1	330	52	st52.sh
380	200	200	1	330	53	st53.sh
410	200	200	1	330	54	st54.sh
460	200	200	1	330	55	st55.sh
470	200	200	1	330	56	st56.sh
500	200	200	1	330	57	st57.sh
540	200	200	1	330	58	st58.sh

MATLAB version

The MATLAB version is found be typing `version` into the command window.

Table E.4: MATLAB version number for simulation

MATLAB version	`'9.2.0.538062 (R2017a)'`

Shell scripts used for simulation

To run the simulations as a batch script with slurm, shell scripts are created that call the relevant MATLAB script. The first script for Packet Delivery Ratio (PDR) simulations are give by Code snippet E.1. The first script for worst case PDR simulations are given by Code snippet E.2. The first script for worst case K2Pruning versus HighConnect is given by Code snippet E.3

Code Snippet E.1: Shell script for PDR simulation, simulatorIndex 61

```
#! /bin/bash
#
#SBATCH --account aauhpc_fat        # partition
#SBATCH --nodes 1                   # number of nodes
#SBATCH --time 24:00:00             # max time (HH:MM:SS)
## SERVER_run(nodeGen,traffic,topology,fairness,dense,startIndex,simulatorIndex)
module load matlab/R2017a
matlab -nodisplay -nosplash -noFigureWindows -r 'SERVER_run(200, 200, 200, 0,1,1,61)'
```

Code Snippet E.2: Shell script for worst case PDR simulation, simulatorIndex 101

```
#! /bin/bash
#
#SBATCH --account aauhpc_fat        # partition
#SBATCH --nodes 1                   # number of nodes
#SBATCH --time 24:00:00             # max time (HH:MM:SS)
## SERVER_run(nodeGen,traffic,topology,fairness,dense,startIndex,simulatorIndex)
module load matlab/R2017a
matlab -nodisplay -nosplash -noFigureWindows -r 'SERVER_run(200, 200, 200, 1,1,1,101)'
```

Code Snippet E.3: Shell script for K2pruning vs CDS simulation, simulatorIndex 11

```
#! /bin/bash
#
#SBATCH --account aauhpc_fat        # partition
#SBATCH --nodes 1                                   # number of nodes
#SBATCH --time 6:00:00              # max time (HH:MM:SS)

# #USAGE SERVER_k2cds(nodeGen,traffic,topology,fairness,len,simulatorIndex)
module load matlab/R2017a
matlab -nodisplay -nosplash -noFigureWindows -r 'SERVER_k2cds(200, 200, 200, 1,60,11)'
```

Matlab scripts used for simulation

The MATLAB scripts that run the simulations for Packet Delivery Ratio (PDR) and worst case PDR are given by Code snippet E.4.

Code Snippet E.4: Matlab script for running PDR and worst case PDR simulations

```
function SERVER_run(nodeGen,traffic,topology,fairness,dense,startIndex,simulatorIndex)
%USAGE SERVER_run(nodeGen,traffic,topology,fairness,dense,startIndex,simulatorIndex)
SNIP_hashfiles

trafficLevel = round(logspace(log10(5),log10(200),12));
folder = 'firstMaySimulator';
mkdir(folder);
seed.nodeGen = nodeGen; %controls node placement and Pathloss generation
seed.enableNodeGen = 1;
seed.traffic = traffic; %controls which seed traffic is generated from
seed.enableTraffic = 1;
seed.topology = topology; %controls which seed the topology is generated form
seed.enableTopology = 1;

relayAlgs = {'k2pruning','highConnect','allRelay','BermanAlgo','FirstCome','none'};

opt.fairness = fairness;%fairness = 1 is worst case
spmd(24)
        s = RandStream('twister'); % Default seed 0.
        RandStream.setGlobalStream(s);
        if any(labindex==1:6)
                TL = trafficLevel(startIndex);
        elseif any(labindex==7:12)
                TL = trafficLevel(startIndex+1);
        elseif any(labindex==13:18)
                TL = trafficLevel(startIndex+2);
        elseif any(labindex==19:24)
                TL = trafficLevel(startIndex+3);
        end
        opt.relayAlg = relayAlgs{mod(labindex-1,6)+1};

        opt.trafficLevel = TL;
        opt.simulatorIndex = simulatorIndex;
        opt.labindex = labindex;
        opt.seeds = seed;
        opt.longSim = 1;
        opt.dense = dense; %set to 1 if dense, 0 if sparse
        %for debugging afterwards
        opt
        options = optFuncNew(opt)
        [nodes, PL,~,~] = simulator(options);
        parsave(sprintf('%s/sim_%d_%d.mat',folder,simulatorIndex,labindex),'nodes',nodes,...
                                        'options',options,'hashes',hashes,'PL',PL);
end
```

The MATLAB script for running comparisons between worst case K2Pruning and HighConnect are given by Code snippet E.5

Code Snippet E.5: MATLAB script for running K2Pruning versus HighConnect

```
function SERVER_k2cds(nodeGen,traffic,topology,fairness,len,simulatorIndex)
%USAGE SERVER_k2cds(nodeGen,traffic,topology,fairness,len,simulatorIndex)

SNIP_hashfiles

trafficLevel = round(logspace(log10(5),log10(200),12));
folder = 'k2cds';
mkdir(folder);
diary(sprintf('%s/diary_%d',folder,simulatorIndex));
seed.nodeGen = nodeGen; %controls node placement and Pathloss generation
seed.enableNodeGen = 1;
seed.traffic = traffic; %controls which seed traffic is generated from
seed.enableTraffic = 1;
seed.topology = topology; %controls which seed the topology is generated form
seed.enableTopology = 1;

relayAlgs = {'k2pruning','highConnect'};

opt.fairness = fairness; %fairness = 1 is worst case
spmd(24)
       s = RandStream('twister'); % Default seed 0.
       RandStream.setGlobalStream(s);
       TL = trafficLevel(ceil(labindex/2)); %labindex is a variable that specifies which
           worker you are in a cluster.
       opt.relayAlg = relayAlgs{mod(labindex-1,2)+1};
       disp(opt.relayAlg)
       opt.trafficLevel = TL;
       opt.simulatorIndex = simulatorIndex;
       opt.labindex = labindex;
       opt.seeds = seed;
       opt.longSim = 1; %specifies that the simulator will run for 10 000 seconds.
       opt.len = len;
       %for debugging afterwards
       opt
       options = optFuncNew(opt)
       [nodes, PL,~,~] = simulator(options);
       parsave(sprintf('%s/sim_%d_%d.mat',folder,simulatorIndex,labindex),'nodes',nodes,...
                                   'options',options,'hashes',hashes,'PL',PL);
end
end
```

Method

Step by step of test, maybe in enumerate

PDR simulation

1. Run `sbatch st61.sh`
2. Run `sbatch st62.sh`
3. Run ...
4. Run `sbatch st78.sh`

Worst case simulation

1. Run `sbatch st101sh`
2. Run `sbatch st102.sh`
3. Run `...`
4. Run `sbatch st118.sh`

K2Pruning versus HighConnect simulation

1. Run `sbatch st11.sh`
2. Run `sbatch st12.sh`
3. Run `...`
4. Run `sbatch st58.sh`

Measurement data

The data which is returned from the simulation can be gotten on request, since the uncompressed data takes up $\approx 62\,\mathrm{GB}$ disk space.

Data processing

PDR simulation

Run script getDataEval.m to extract data from savefiles
Run script plotDataEval.m to plot the data.

Worst case PDR simulation

Run script getData_fairness.m to extract data from savefiles
Run script plotData_fairness.m to plot the data.

K2Pruning versus HighConnect PDR simulation

Run script getData_kcds.m to extract data from savefiles
Run script plotData_k2cds.m to plot the data.

Simulation journal: Comparison of implemented and theoretical algorithm

F

Date: 05/05-2017

Purpose

The purpose of this simulation is to test how well the implementation of the highest connectivity compares to the ideal. This is done with 200 nodes in a square 60 m by 60 m area. First, the implementation of the algorithm creates a relay set dynamically. Then the relay indices are saved to an external folder. Lastly, the relays made with implementation and ideal are tested on a range of different seeds[1].

Simulation parameters

In Table F.1, the parameters used to setup the implemented algorithm is given. **Nodegen** specifies which seed is used for generating the nodeset. **Len** specifies which area is used, 60 for 60 m by 60 m area. **Labindex** gives which worker the simulation is done on. **SimIndex** specifies which simulator index is used, this decides where the data is saved, removing the possibility of overwriting other simulations. **Script** specifies which script is used to run the simulation.

[1] The Mersenne Twister random number generator is used for all random numbers.

Table F.1: Specifications for the implementation.

Nodegen	**Len**	**LabIndex**	**SimIndex**	**Script**
200	60	1	3	dyn3.sh
230	60	2	3	dyn3.sh
260	60	3	3	dyn3.sh
290	60	4	3	dyn3.sh
320	60	5	3	dyn3.sh
360	60	6	3	dyn3.sh
380	60	7	3	dyn3.sh
410	60	8	3	dyn3.sh
460	60	9	3	dyn3.sh
470	60	10	3	dyn3.sh
500	60	11	3	dyn3.sh
540	60	12	3	dyn3.sh
560	60	13	3	dyn3.sh
580	60	14	3	dyn3.sh
600	60	15	3	dyn3.sh
620	60	16	3	dyn3.sh
640	60	17	3	dyn3.sh
660	60	18	3	dyn3.sh
680	60	19	3	dyn3.sh
700	60	20	3	dyn3.sh
720	60	21	3	dyn3.sh
740	60	22	3	dyn3.sh
760	60	23	3	dyn3.sh
780	60	24	3	dyn3.sh

The parameters in Table F.2 specifies how the implementation and the theoretical is simulated to compare the two algorithms. **Nodegen** specifies which seed is used to generate the nodeset. **Traffic** specifies the seed to use when creating timing, who sends, and who receives. **Fairness** specifies if extra packets are sent from the nodes furthest away from each other. **Length** specifies which area is used for the simulation, 60 for 60 m by 60 m area. **SimIndex** specifies which simulator index is used, and is used to specify where the simulator saves files. **EndMsg** specifies how many messages are sent before the simulator saves and exits the simulation. **trafficlevel** gives the exact traffic level intensites that are simulated. Here, ls() means logspace, and creates logarithmically spaced numbers. **Script** specifies which shell script runs the simulation.

Table F.2: Specifications for the implementation.

Nodegen	Traffic	Fairness	Length	SimIndex	EndMsg	trafficlevel	Script
200	200	0	60	401	1000	ls(log10(5),log10(200),12)	st401.sh
230	200	0	60	402	1000	ls(log10(5),log10(200),12)	st402.sh
260	200	0	60	403	1000	ls(log10(5),log10(200),12)	st403.sh
290	200	0	60	404	1000	ls(log10(5),log10(200),12)	st404.sh
320	200	0	60	405	1000	ls(log10(5),log10(200),12)	st405.sh
360	200	0	60	406	1000	ls(log10(5),log10(200),12)	st406.sh
380	200	0	60	407	1000	ls(log10(5),log10(200),12)	st407.sh
410	200	0	60	408	1000	ls(log10(5),log10(200),12)	st408.sh
460	200	0	60	409	1000	ls(log10(5),log10(200),12)	st409.sh
470	200	0	60	410	1000	ls(log10(5),log10(200),12)	st410.sh
500	200	0	60	411	1000	ls(log10(5),log10(200),12)	st411.sh
540	200	0	60	412	1000	ls(log10(5),log10(200),12)	st412.sh
560	200	0	60	413	1000	ls(log10(5),log10(200),12)	st413.sh
580	200	0	60	414	1000	ls(log10(5),log10(200),12)	st414.sh
600	200	0	60	415	1000	ls(log10(5),log10(200),12)	st415.sh
620	200	0	60	416	1000	ls(log10(5),log10(200),12)	st416.sh
640	200	0	60	417	1000	ls(log10(5),log10(200),12)	st417.sh
660	200	0	60	418	1000	ls(log10(5),log10(200),12)	st418.sh
680	200	0	60	419	1000	ls(log10(5),log10(200),12)	st419.sh
700	200	0	60	420	1000	ls(log10(5),log10(200),12)	st420.sh
720	200	0	60	421	1000	ls(log10(5),log10(200),12)	st421.sh
740	200	0	60	422	1000	ls(log10(5),log10(200),12)	st422.sh
760	200	0	60	423	1000	ls(log10(5),log10(200),12)	st423.sh
780	200	0	60	424	1000	ls(log10(5),log10(200),12)	st424.sh

MATLAB version

The MATLAB version is found be typing `version` into the command window.

Table F.3: MATLAB version number for simulation

MATLAB version	`'9.2.0.538062 (R2017a)'`

Shell scripts used for simulation

To run the simulations as a batch script with slurm, shell scripts are made that call the MATLAB script. Code snippet F.1 calls the setup of the implementation. Code snippet F.2 calls MATLAB script with simulatorIndex 401.

Code Snippet F.1: Shell script for setting up implementation, script name dyn3.sh

```
1 #! /bin/bash
2 #
3 #SBATCH --account aauhpc_fat        # number of nodes
4 #SBATCH --nodes 1
```

```
#SBATCH --time 24:00:00                    # max time (HH:MM:SS)
#SBATCH --mail-type=ALL
#SBATCH --mail-user=eajh14@student.aau.dk
#USAGE SERVER_dynSetup200(simulatorIndex)

module load matlab/R2017a
matlab -nodisplay -nosplash -noFigureWindows -r 'SERVER_dynSetup200(3)'
```

Code Snippet F.2: Shell script for creating data to compare implementation and theoretical.

```
#! /bin/bash
#
#SBATCH --account aauhpc_slim       # number of nodes
#SBATCH --nodes 1                   # number of nodes
#SBATCH --time 4:00:00              # max time (HH:MM:SS)
#SBATCH --mail-type=ALL
#SBATCH --mail-user=eajh14@student.aau.dk

#USAGE SERVER_stCDS200(simulatorIndex)
matlab -nodisplay -nosplash -noFigureWindows -r 'SERVER_stCDS200(401)'
```

Matlab scripts used for simulation

The MATLAB scripts that run the simulations for setup of implementation is given by Code snippet F.3

Code Snippet F.3: Matlab script setting up implementation

```
function SERVER_dynSetup200(simulatorIndex)
        trafficLevel = round(logspace(log10(5),log10(200),12));
        folder = 'dynamic_setup';
        mkdir(folder);
        seed.nodeGen = 0; %controls node placement and Pathloss generation
        seed.enableNodeGen = 1;
        seed.traffic = 200; %controls which seed traffic is generated from
        seed.enableTraffic = 1;
        seed.topology = 100; %controls which seed the topology is generated form
        seed.enableTopology = 1;

        relayAlg = 'none';
        try
                parpool(24)
        catch
                disp('no parpool created, perhaps one is already running.')
        end
        c = gcp
        if c.NumWorkers ~= 24
                error('Parpool was not created succesfully');
        end

        opt.fairness = 0;
        spmd(24)
                s = RandStream('twister'); % Default seed 0.
```

```
26                  RandStream.setGlobalStream(s);
27                  switch labindex %labindex returns which worker the code is on
28                  case 1
29                         seed.nodeGen = 200;
30                         len = 60;
31                  case 2
32                         seed.nodeGen = 230;
33                         len = 60;
34                  case 3
35                         seed.nodeGen = 260;
36                         len = 60;
37                  case 4
38                         seed.nodeGen = 290;
39                         len = 60;
40                  case 5
41                         seed.nodeGen = 320;
42                         len = 60;
43                  case 6
44                         seed.nodeGen = 360;
45                         len = 60;
46                  case 7
47                         seed.nodeGen = 380;
48                         len = 60;
49                  case 8
50                         seed.nodeGen = 410;
51                         len = 60;
52                  case 9
53                         seed.nodeGen = 460;
54                         len = 60;
55                  case 10
56                         seed.nodeGen = 470;
57                         len = 60;
58                  case 11
59                         seed.nodeGen = 500;
60                         len = 60;
61                  case 12
62                         seed.nodeGen = 540;
63                         len = 60;
64                  case 13
65                         seed.nodeGen = 560;
66                         len = 60;
67                  case 14
68                         seed.nodeGen = 580;
69                         len = 60;
70                  case 15
71                         seed.nodeGen = 600;
72                         len = 60;
73                  case 16
74                         seed.nodeGen = 620;
75                         len = 60;
76                  case 17
77                         seed.nodeGen = 640;
78                         len = 60;
79                  case 18
80                         seed.nodeGen = 660;
81                         len = 60;
82                  case 19
```

```
                        seed.nodeGen = 680;
                        len = 60;
                case 20
                        seed.nodeGen = 700;
                        len = 60;
                case 21
                        seed.nodeGen = 720;
                        len = 60;
                case 22
                        seed.nodeGen = 740;
                        len = 60;
                case 23
                        seed.nodeGen = 760;
                        len = 60;
                case 24
                        seed.nodeGen = 780;
                        len = 60;
                end
                %debugging
                fprintf('lab - %d, nodegen - %d, len - %d',labindex,seed.nodeGen,len);
                opt.relayAlg = 'none'; %no relays in the start
                disp(opt.relayAlg)
                opt.trafficLevel = 1;
                opt.simulatorIndex = simulatorIndex;
                opt.labindex = labindex;
                opt.seeds = seed;
                opt.longSim = 1;
                opt.len = len;
                opt.amount = 200;
                opt.static = 0;
                %for debugging afterwards
                opt
                options = optFuncNew(opt)
                [nodes, PL,~,~] = simulator(options);

        end
end
```

The MATLAB script that collects data for comparisons between implementation and theoretical is given by Code snippet F.4

Code Snippet F.4: Matlab script for running comparison between implementation and theoretical

```
function SERVER_stCDS200(simulatorIndex)
        %run some
        %USAGE SERVER_stCDS200(simulatorIndex)

        SNIP_hashfiles
        opt.endMsg = 1000; %return when 1000 messages have been sent

        trafficLevel = round(logspace(log10(5),log10(200),12));

        seed.nodeGen = 0; %controls node placement and Pathloss generation
        seed.enableNodeGen = 1;
        seed.traffic = 200; %controls which seed traffic is generated from
        seed.enableTraffic = 1;
```

```
14          seed.topology = 100; %controls which seed the topology is generated form
15          seed.enableTopology = 1;
16          opt.len = 60;
17          opt.relayAlg = 'highConnect';
18          try
19                  parpool(24)
20          catch
21                  disp('no parpool created, perhaps one is already running.')
22          end
23          c = gcp
24          if c.NumWorkers ~= 24
25                  error('Parpool was not created succesfully');
26          end
27
28          switch simulatorIndex
29                  case 401
30                          seed.nodeGen = 200;
31                  case 402
32                          seed.nodeGen = 230;
33                  case 403
34                          seed.nodeGen = 260;
35                  case 404
36                          seed.nodeGen = 290;
37                  case 405
38                          seed.nodeGen = 320;
39                  case 406
40                          seed.nodeGen = 360;
41                  case 407
42                          seed.nodeGen = 380;
43                  case 408
44                          seed.nodeGen = 410;
45                  case 409
46                          seed.nodeGen = 460;
47                  case 410
48                          seed.nodeGen = 470;
49                  case 411
50                          seed.nodeGen = 500;
51                  case 412
52                          seed.nodeGen = 540;
53                  case 413
54                          seed.nodeGen = 560;
55                  case 414
56                          seed.nodeGen = 580;
57                  case 415
58                          seed.nodeGen = 600;
59                  case 416
60                          seed.nodeGen = 620;
61                  case 417
62                          seed.nodeGen = 640;
63                  case 418
64                          seed.nodeGen = 660;
65                  case 419
66                          seed.nodeGen = 680;
67                  case 420
68                          seed.nodeGen = 700;
69                  case 421
70                          seed.nodeGen = 720;
```

```
71                case 422
72                       seed.nodeGen = 740;
73                case 423
74                       seed.nodeGen = 760;
75                case 424
76                       seed.nodeGen = 780;
77        end
78        opt.fairness = 0;
79        spmd(24)
80                s = RandStream('twister'); % Default seed 0.
81                RandStream.setGlobalStream(s);
82
83                switch labindex %labindex is the worker number.
84                       case 1
85                              opt.trafficLevel = 5;
86                       case 2
87                              opt.trafficLevel = 7;
88                       case 3
89                              opt.trafficLevel = 10;
90                       case 4
91                              opt.trafficLevel = 14;
92                       case 5
93                              opt.trafficLevel = 19;
94                       case 6
95                              opt.trafficLevel = 27;
96                       case 7
97                              opt.trafficLevel = 37;
98                       case 8
99                              opt.trafficLevel = 52;
100                      case 9
101                             opt.trafficLevel = 73;
102                      case 10
103                             opt.trafficLevel = 102;
104                      case 11
105                             opt.trafficLevel = 143;
106                      case 12
107                             opt.trafficLevel = 200;
108                      case 13
109                             opt.trafficLevel = 5;
110                      case 14
111                             opt.trafficLevel = 7;
112                      case 15
113                             opt.trafficLevel = 10;
114                      case 16
115                             opt.trafficLevel = 14;
116                      case 17
117                             opt.trafficLevel = 19;
118                      case 18
119                             opt.trafficLevel = 27;
120                      case 19
121                             opt.trafficLevel = 37;
122                      case 20
123                             opt.trafficLevel = 52;
124                      case 21
125                             opt.trafficLevel = 73;
126                      case 22
127                             opt.trafficLevel = 102;
```

```
128                         case 23
129                                 opt.trafficLevel = 143;
130                         case 24
131                                 opt.trafficLevel = 200;
132                 end
133                 if labindex >=13
134                 %dynamic
135                 opt.afterDyn = 1;
136                 opt.relayAlg = 'customPath'
137
138                 opt.relpath = 'DynRelay/rel_lst_';
139                 end
140
141                 fprintf('lab␣-␣%d,␣nodegen␣-␣%d,␣len␣-␣%d',labindex,seed.nodeGen,opt.len);
142
143                 disp(opt.relayAlg)
144                 opt.simulatorIndex = simulatorIndex;
145                 opt.labindex = labindex;
146                 opt.seeds = seed;
147                 opt.longSim = 1;
148                 opt.amount = 200;
149                 opt.static = 1;
150                 %for debugging afterwards
151                 opt
152                 options = optFuncNew(opt)
153                 [nodes, PL,~,~] = simulator(options);
154         end
155 end
```

Method

Implementation setup

1. Run shell script `dyn3.sh`.
2. Download folder `/saves3_`.
3. In MATLAB, add `CONV_dyn2relayset.m` to path.
4. In MATLAB, navigate to folder `/saves3_` which was downloaded.
5. Run MATLAB script `CONV_dyn2relayset.m`.
6. Copy the resultant set to folder `/simulator/DynRelay`, where `/simulator` is the folder in which `simulator.m` is.
7. Upload folder `/DynRelay`.

Comparison

1. Run shell scripts `st401.sh, 402.sh, ... st424.sh`.
2. When the simulation is done, download folders `/saves401_` to `/saves424_`.

Measurement data

Data can be gotten on request.

Data processing

1. Navigate to the folder in which folders `/saves401_` to `/saves424_` exists.
2. Run MATLAB script `getData_200.m`.
3. Run MATLAB script `SNIP_plot_latex.m` to plot data.

Simulation journal: Optimised PPS function

G

Date: 11/05-2017

Purpose

The purpose of this simulation is to determine a function description the optimal Packets Per Second (PPS) dependent on the amount of neighbours a node has.

Simulation

Firstly $10 \cdot 100 = 1000$ simulations are run by executing the MATLAB code seen in Code snippet G.1. In Code snippet G.1 the function `find_PPS_function_options` is used, this function `find_PPS_function_options` is seen in Code snippet G.2.

Code Snippet G.1: Code used to generate PDR for different amount of nodes and different PPS settings.

```
PPS = 0.1:0.1:10;
run = 100:100:1000;
ResPDR = cell([1 numel(run)]);
parfor m = 1:numel(run) % Run entry
    options = find_PPS_function_optons( run(m) )
    PDR = []; % zeros(1,options.amountNodes);
    for n = PPS

        options.timeInSeconds = (options.numMessages)/(n * run(m));

        [nodes,~,~,~] = simulator(options); % Run a simulation

        % Calculate the PDR
        rec = zeros(1,options.amountNodes);
        trans = zeros(1,options.amountNodes);
        for a = 1:options.amountNodes
            rec(a) = nodes{a}.statReceived;
            trans(a) = nodes{a}.statTransmission;
        end
        % Find average
        recHat = sum(rec)/numel(rec);
        transHat = sum(trans)/numel(trans);
```

```
        PDR(end+1) = recHat/(sum(trans)-transHat);
    end

    % Save found PDR
    ResPDR{m} = PDR;
end
```

Code Snippet G.2: The option function used in Code snippet G.1.

```
function options = find_PPS_function_options(m)
options = optFunc('baseline100',100,1);

options.relayAlg = 'none';
options.lenX = 1;
options.lenY = 1;
options.rngSeed = 200;
options.globalTTL = 1;

options.reuseNodefile = 0;
options.reusePathLossModel = 0;

options.rngDefault = 1;
options.staticTest = 1;

options.verbose = 0;
options.statusVerbose = 0;

% Main parameter
options.numMessages = 500;
options.amountNodes = m;
end
```

Data processing

The results from running the code in Code snippet G.1 is processed by running the code seen in Code snippet G.3.

Code Snippet G.3: Code used to precess the results from Code snippet G.1.

```
syms cost
OptimalPPS = [];
OptimalPBD = [];

ResFitPDR = {};
for n = 1:numel(run) % = options.amountNodes
    f = fit(PPS',ResPDR{n}','poly5');
    fitPDR = f.p1 * PPS.^5 + f.p2*PPS.^4 + f.p3*PPS.^3 + f.p4*PPS.^2 + f.p5*PPS + f.p6;

    % Calculate optimal PPS for min cost
    numPBD = 5; % I will be seen later that this is enough
    PBD = 1:1:numPBD;

    A = zeros(numPBD,numel(PPS));
    OV = ones(1,numel(PPS))*0.99999;
```

```
16
17      temp = min([ fitPDR ; OV]); % PDR abouve 1 it will be set to 0.99999
18      A = 1-(1 - temp ).^(PBD'); % Find PDR
19      B = A >= 0.95; % Find PDR > 0.95
20      B =  B+(B==0) * 100000000; % If PDR is below 0.95 cost is set high
21      cost = PBD' ./ PPS; % Find cost
22      C = B .* cost; % Weighted cost
23      [~,ind] = min(C(:)); % Find entry in C which has lowest value
24      [i,m] = ind2sub(size(C),ind); % Translate entry to row and column
25
26      % Save res fit
27      ResFitPDR{end+1} = fitPDR;
28
29      % Save Optimal PPS and PBD
30      OptimalPPS(end+1) = PPS(m);
31      OptimalPBD(end+1) = PBD(i);
32  end
```

From Code snippet G.1 it is seen how a cost function, $cost = \frac{PBD}{PPS}$, Is defined. In this cost function `PBD` is the amount of packet a node have to send before it with a 95% probability can be said that a neighbour node have received at least one of these send packets.

MATLAB version

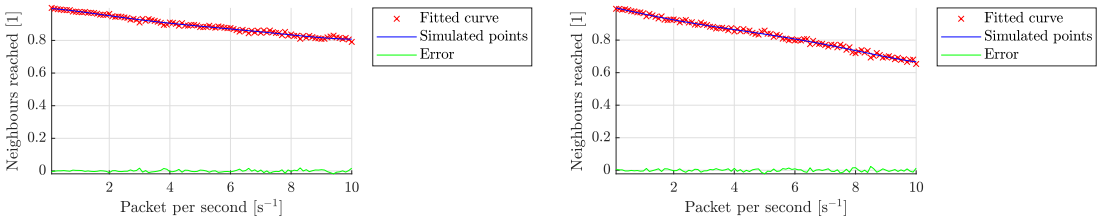
The MATLAB version is found be typing `version` into the command window.

Table G.1: MATLAB version number for simulation

MATLAB version	'9.2.0.538062 (R2017a)')

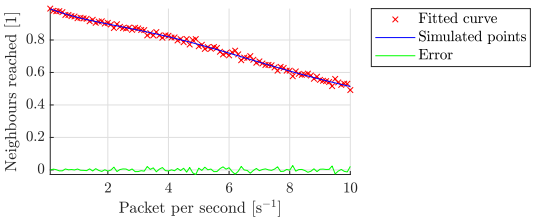
Results

On figure Figure G.1 to Figure G.7 the results are seen.

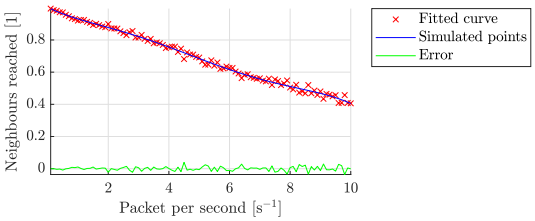


(a) Curve fit for 100 neighbours. ***(b)*** Curve fit for 200 neighbours.

Figure G.1: Two plots showing the found curve fits.

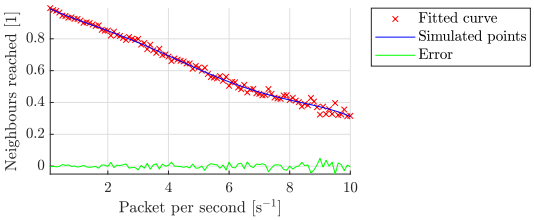


(a) Curve fit for 300 neighbours.

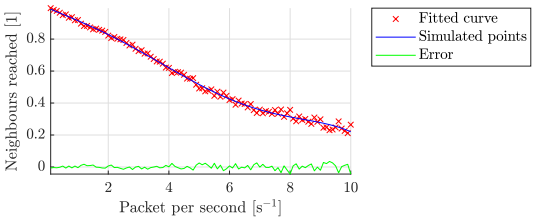


(b) Curve fit for 400 neighbours.

Figure G.2: Two plots showing the found curve fits.

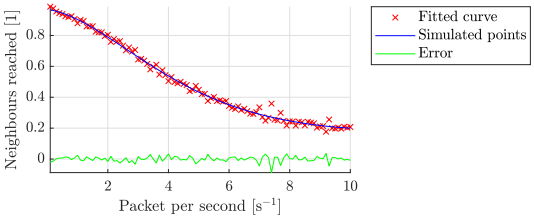


(a) Curve fit for 500 neighbours.

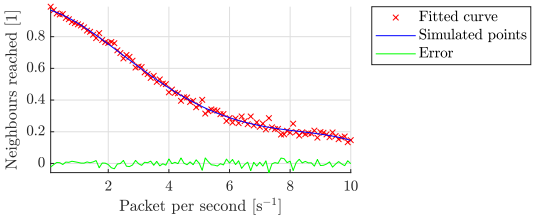


(b) Curve fit for 600 neighbours.

Figure G.3: Two plots showing the found curve fits.

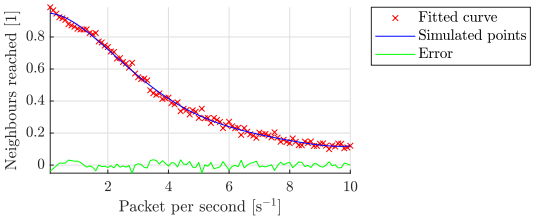


(a) Curve fit for 700 neighbours.

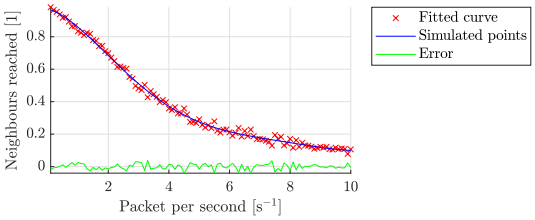


(b) Curve fit for 800 neighbours.

Figure G.4: Two plots showing the found curve fits.



(a) Curve fit for 900 neighbours.



(b) Curve fit for 1000 neighbours.

Figure G.5: Two plots showing the found curve fits.

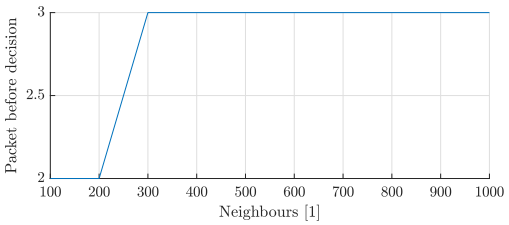
Conclusion

A curve is fitted to the plot seen on Figure G.5a with the code seen in Code snippet G.4.

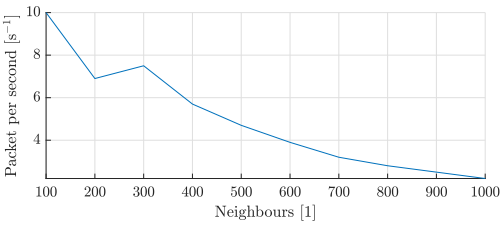
Code Snippet G.4: Code which do a curvefit for OptimalPPS.

```
1    fPPS = fit(run',OptimalPPS',Exp1');
2    FitPPS = fPPS.a*exp(fPPS.b*run);
```

On Figure G.8 the final curve fit is seen.

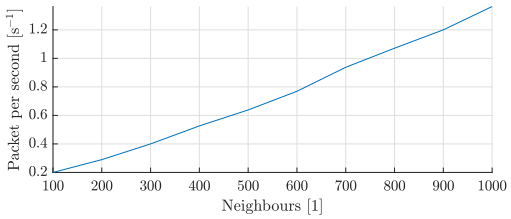


(a) Plot showing the matching PBD to the PPS graph seen on Figure G.6b.

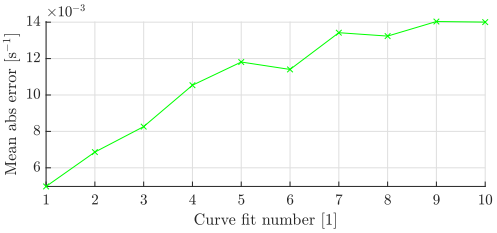


(b) Optimised PPS value dependent on the PPS.

Figure G.6: Plot showing the Packets Per Second (PPS) and PBD resulting from the minimisation.



(a) Plot showing the Cost.



(b) Error of the all the curve fits.

Figure G.7: Plots showing the time cost and the error of the curve fits.

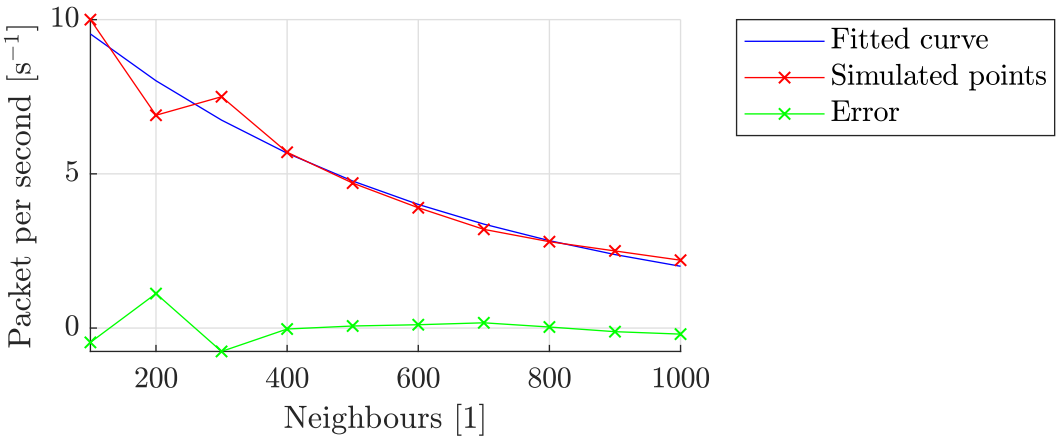


Figure G.8: Graph showing the curve fit of the optimised Packets Per Second (PPS) function.

The curve on Figure G.8 is as described in Equation (G.1).

$$\texttt{PPS} = 11.3405 \cdot \exp(-0.0017334 \cdot \text{Neighbours}) \tag{G.1}$$

Furthermore it is seen from Figure G.6a that with more than 200 neighbours. A node needs to send three packets before its neighbours have a 95% probability of receiving at least one of the packets.

Test data: Acceptance simulation

H

Table H.1: Data for implementation Packet Delivery Ratio

seed	traffic level											
	5	7	10	14	19	27	37	52	73	102	142	200
200	1.00	1.00	1.00	1.00	1.00	0.99	0.99	0.94	0.86	0.75	0.71	0.57
230	1.00	1.00	1.00	0.99	0.99	0.98	0.98	0.95	0.86	0.77	0.69	0.61
260	1.00	1.00	1.00	1.00	1.00	0.99	0.99	0.94	0.86	0.77	0.69	0.60
290	1.00	0.99	0.99	0.99	0.99	0.99	0.99	0.94	0.86	0.72	0.66	0.58
320	1.00	1.00	1.00	0.99	0.99	0.98	0.97	0.95	0.86	0.77	0.72	0.60
360	1.00	1.00	0.99	1.00	0.99	0.99	0.98	0.92	0.87	0.76	0.68	0.58
380	1.00	1.00	1.00	1.00	0.99	0.99	0.98	0.94	0.88	0.78	0.71	0.60
410	0.99	0.99	0.99	0.99	0.98	0.97	0.96	0.94	0.83	0.73	0.65	0.56
460	1.00	1.00	1.00	1.00	0.99	0.99	0.99	0.95	0.88	0.78	0.70	0.64
470	1.00	1.00	0.99	0.99	1.00	0.99	0.99	0.95	0.86	0.76	0.69	0.59
500	1.00	1.00	1.00	0.99	1.00	0.99	0.99	0.94	0.87	0.80	0.72	0.60
540	1.00	1.00	0.99	0.99	0.99	0.99	0.98	0.94	0.86	0.75	0.71	0.60
560	1.00	1.00	1.00	1.00	1.00	0.99	0.98	0.95	0.85	0.75	0.70	0.58
580	1.00	1.00	1.00	1.00	1.00	0.99	0.98	0.94	0.87	0.80	0.71	0.60
600	1.00	1.00	1.00	1.00	1.00	0.99	0.99	0.94	0.85	0.76	0.68	0.60
620	0.99	1.00	1.00	0.99	0.99	0.99	0.99	0.94	0.88	0.78	0.71	0.61
640	1.00	1.00	1.00	1.00	1.00	0.99	0.98	0.94	0.86	0.74	0.66	0.56
660	1.00	1.00	1.00	1.00	0.99	0.99	0.97	0.94	0.86	0.76	0.69	0.59
680	1.00	1.00	0.99	0.99	0.99	0.99	0.99	0.94	0.86	0.76	0.68	0.56
700	1.00	1.00	1.00	1.00	0.99	0.99	0.99	0.94	0.86	0.73	0.68	0.60
720	1.00	1.00	1.00	1.00	1.00	1.00	0.99	0.94	0.87	0.74	0.71	0.59
740	1.00	1.00	1.00	0.99	1.00	0.99	0.99	0.96	0.88	0.78	0.70	0.62
760	1.00	1.00	1.00	1.00	1.00	0.99	0.99	0.94	0.87	0.78	0.68	0.57
780	1.00	0.99	0.99	0.99	0.99	0.98	0.99	0.96	0.86	0.75	0.67	0.58

Table H.2: Data for theoretical Packet Delivery Ratio

Seed	Traffic level											
	5	7	10	14	19	27	37	52	73	102	143	200
200	0.99	0.99	0.99	0.99	0.99	0.98	0.97	0.94	0.84	0.76	0.69	0.59
230	1.00	1.00	1.00	1.00	0.99	0.98	0.98	0.95	0.88	0.77	0.71	0.61
260	1.00	1.00	0.99	1.00	0.98	0.99	0.98	0.93	0.86	0.77	0.68	0.60
290	1.00	1.00	1.00	0.99	0.99	0.99	0.98	0.94	0.85	0.77	0.68	0.60
320	1.00	0.99	0.99	0.99	0.99	0.98	0.97	0.94	0.87	0.78	0.72	0.63
360	1.00	1.00	1.00	1.00	0.99	0.99	0.99	0.94	0.86	0.77	0.68	0.59
380	1.00	1.00	0.99	1.00	0.99	0.99	0.98	0.94	0.86	0.78	0.69	0.61
410	1.00	0.99	0.99	0.99	0.99	0.99	0.98	0.95	0.86	0.75	0.67	0.59
460	1.00	1.00	1.00	1.00	1.00	0.99	0.99	0.95	0.87	0.78	0.71	0.61
470	1.00	0.99	0.99	0.99	0.99	0.98	0.98	0.95	0.86	0.77	0.69	0.61
500	1.00	1.00	1.00	1.00	0.99	0.98	0.97	0.94	0.87	0.76	0.72	0.61
540	1.00	0.99	0.99	0.99	0.98	0.98	0.96	0.93	0.83	0.74	0.67	0.58
560	1.00	1.00	1.00	0.99	1.00	0.99	0.99	0.94	0.89	0.79	0.72	0.60
580	1.00	0.99	1.00	0.99	0.99	0.98	0.98	0.94	0.85	0.77	0.70	0.61
600	0.99	0.99	0.99	0.99	0.99	0.98	0.97	0.94	0.85	0.75	0.67	0.60
620	0.99	1.00	0.98	0.99	0.98	0.97	0.97	0.93	0.86	0.75	0.69	0.59
640	1.00	1.00	1.00	0.99	0.99	0.99	0.98	0.94	0.86	0.77	0.66	0.56
660	1.00	1.00	1.00	0.99	0.99	0.99	0.98	0.94	0.85	0.75	0.67	0.56
680	1.00	1.00	0.99	0.99	0.99	0.99	0.99	0.95	0.87	0.75	0.69	0.60
700	1.00	1.00	0.99	1.00	0.99	0.98	0.98	0.94	0.85	0.72	0.69	0.58
720	1.00	0.99	1.00	1.00	1.00	0.99	0.98	0.95	0.86	0.77	0.68	0.60
740	1.00	1.00	0.99	0.98	0.99	0.98	0.99	0.96	0.86	0.76	0.70	0.62
760	0.98	0.99	0.98	0.98	0.97	0.97	0.97	0.92	0.82	0.74	0.68	0.58
780	1.00	1.00	0.99	0.99	1.00	0.99	0.98	0.95	0.85	0.75	0.68	0.58

Table H.3: Data for implementation throughput

seed	traffic level											
	5	7	10	14	19	27	37	52	73	102	142	200
200	0.23	0.33	0.48	0.66	0.90	1.25	1.71	2.34	2.93	3.71	4.55	5.57
230	0.23	0.33	0.49	0.65	0.89	1.23	1.69	2.37	2.94	3.80	4.43	6.03
260	0.23	0.33	0.49	0.65	0.90	1.24	1.72	2.35	2.93	3.82	4.38	5.92
290	0.23	0.33	0.48	0.65	0.90	1.24	1.71	2.34	2.93	3.56	4.24	5.70
320	0.23	0.33	0.48	0.65	0.89	1.23	1.68	2.37	2.93	3.82	4.57	5.92
360	0.23	0.33	0.48	0.65	0.90	1.24	1.70	2.31	2.97	3.76	4.34	5.68
380	0.23	0.33	0.49	0.65	0.90	1.24	1.70	2.35	2.99	3.87	4.57	5.91
410	0.23	0.33	0.48	0.65	0.89	1.22	1.66	2.34	2.82	3.60	4.16	5.50
460	0.23	0.33	0.48	0.65	0.90	1.24	1.71	2.38	3.00	3.85	4.47	6.23
470	0.23	0.33	0.48	0.65	0.90	1.25	1.71	2.38	2.94	3.74	4.39	5.83
500	0.23	0.33	0.49	0.65	0.90	1.24	1.72	2.34	2.96	3.92	4.62	5.87
540	0.23	0.33	0.48	0.65	0.90	1.24	1.70	2.34	2.93	3.69	4.54	5.93
560	0.23	0.33	0.49	0.66	0.90	1.24	1.70	2.37	2.91	3.68	4.45	5.70
580	0.23	0.33	0.49	0.66	0.90	1.24	1.70	2.34	2.97	3.93	4.55	5.90
600	0.23	0.33	0.48	0.65	0.90	1.25	1.72	2.35	2.88	3.73	4.33	5.86
620	0.23	0.33	0.48	0.65	0.90	1.24	1.71	2.36	3.00	3.85	4.51	6.02
640	0.23	0.33	0.48	0.65	0.90	1.25	1.70	2.35	2.93	3.65	4.19	5.49
660	0.23	0.33	0.48	0.65	0.89	1.24	1.68	2.36	2.94	3.74	4.40	5.81
680	0.23	0.33	0.48	0.65	0.89	1.24	1.71	2.34	2.94	3.75	4.36	5.53
700	0.22	0.34	0.47	0.63	0.86	1.24	1.72	2.36	2.95	3.60	4.36	5.90
720	0.23	0.33	0.49	0.65	0.90	1.25	1.72	2.36	2.97	3.66	4.52	5.78
740	0.23	0.33	0.48	0.65	0.90	1.24	1.71	2.39	2.99	3.87	4.47	6.12
760	0.23	0.33	0.48	0.65	0.90	1.24	1.71	2.35	2.95	3.83	4.33	5.61
780	0.23	0.33	0.48	0.65	0.90	1.23	1.71	2.39	2.94	3.71	4.28	5.72

Table H.4: Data for theoretical throughput

Seed	Traffic level											
	5	7	10	14	19	27	37	52	73	102	143	200
200	0.23	0.33	0.48	0.65	0.89	1.23	1.68	2.35	2.87	3.76	4.48	5.82
230	0.23	0.33	0.48	0.65	0.90	1.23	1.70	2.36	3.01	3.79	4.61	5.98
260	0.23	0.33	0.48	0.65	0.89	1.24	1.69	2.33	2.93	3.78	4.41	5.94
290	0.23	0.33	0.48	0.65	0.89	1.24	1.69	2.34	2.88	3.79	4.42	5.87
320	0.23	0.33	0.48	0.65	0.89	1.24	1.68	2.36	2.96	3.85	4.65	6.17
360	0.23	0.33	0.48	0.65	0.90	1.24	1.72	2.35	2.92	3.78	4.38	5.79
380	0.23	0.33	0.48	0.65	0.90	1.24	1.70	2.36	2.93	3.83	4.45	5.99
410	0.23	0.33	0.48	0.65	0.90	1.24	1.70	2.37	2.92	3.69	4.34	5.80
460	0.23	0.33	0.48	0.66	0.90	1.24	1.71	2.38	2.96	3.87	4.59	5.99
470	0.23	0.33	0.48	0.65	0.89	1.23	1.69	2.36	2.94	3.79	4.47	6.00
500	0.23	0.33	0.48	0.65	0.89	1.23	1.68	2.34	2.96	3.77	4.67	5.96
540	0.23	0.33	0.48	0.65	0.89	1.24	1.67	2.32	2.83	3.63	4.32	5.66
560	0.23	0.33	0.49	0.65	0.90	1.25	1.71	2.35	3.04	3.88	4.64	5.86
580	0.23	0.33	0.48	0.65	0.89	1.22	1.69	2.34	2.91	3.79	4.56	6.03
600	0.23	0.33	0.48	0.65	0.89	1.23	1.68	2.33	2.89	3.70	4.37	5.86
620	0.23	0.33	0.48	0.65	0.89	1.22	1.68	2.33	2.92	3.71	4.46	5.83
640	0.23	0.33	0.48	0.65	0.90	1.24	1.69	2.34	2.93	3.79	4.26	5.54
660	0.23	0.33	0.48	0.65	0.90	1.24	1.70	2.36	2.91	3.71	4.31	5.54
680	0.24	0.33	0.48	0.65	0.89	1.24	1.71	2.38	2.96	3.68	4.48	5.85
700	0.23	0.34	0.47	0.65	0.87	1.24	1.70	2.36	2.89	3.56	4.45	5.72
720	0.23	0.33	0.49	0.65	0.90	1.24	1.70	2.37	2.94	3.79	4.38	5.94
740	0.23	0.33	0.48	0.65	0.90	1.23	1.71	2.39	2.94	3.76	4.52	6.06
760	0.23	0.33	0.48	0.64	0.88	1.22	1.67	2.30	2.81	3.66	4.38	5.70
780	0.23	0.33	0.48	0.65	0.90	1.24	1.70	2.37	2.90	3.72	4.43	5.68

Table H.5: Data for implementation latency

seed	traffic level											
	5	7	10	14	19	27	37	52	73	102	142	200
200	11.37	12.50	13.42	14.39	17.29	23.71	51.32	175.20	242.93	217.11	203.71	143.67
230	10.50	11.28	11.65	13.00	14.36	18.24	30.92	101.06	167.95	168.45	154.93	101.10
260	10.24	11.31	12.18	13.47	16.22	22.21	46.45	141.15	205.73	207.07	211.72	146.79
290	11.48	12.65	13.65	14.23	16.64	22.58	38.27	137.94	204.99	212.74	201.56	150.02
320	10.39	11.60	11.60	12.85	14.92	19.32	29.09	101.95	160.12	166.52	129.46	104.60
360	11.29	13.19	13.93	14.82	17.50	23.85	49.89	156.82	215.12	250.39	183.77	133.43
380	10.75	11.87	12.32	13.72	15.36	19.83	37.21	137.83	197.21	202.56	172.93	128.09
410	14.05	14.79	14.70	17.02	18.66	22.24	33.04	102.51	185.22	179.56	187.76	121.31
460	10.45	11.13	12.31	13.10	14.76	19.02	37.61	115.88	173.06	185.13	161.43	133.50
470	11.59	11.38	12.80	14.12	17.45	20.84	44.50	139.72	202.98	195.23	179.85	126.44
500	11.00	11.33	11.78	12.57	15.12	19.42	37.33	119.78	183.66	197.17	165.91	121.65
540	11.65	12.88	13.33	14.36	15.89	21.02	41.87	143.98	202.50	203.71	175.50	135.41
560	11.15	11.14	13.26	13.12	16.76	21.78	38.10	149.14	205.54	212.25	179.87	130.51
580	10.61	10.94	12.00	13.32	16.03	21.67	43.92	137.12	209.96	218.43	188.81	142.04
600	10.58	11.71	11.35	13.06	15.62	19.05	38.65	119.38	174.95	186.07	176.28	126.04
620	10.58	11.78	12.13	12.89	15.40	20.67	34.01	110.39	171.95	200.57	156.08	126.52
640	13.09	13.65	13.81	16.59	19.87	25.14	48.49	175.24	217.32	222.89	208.14	147.37
660	11.99	13.17	12.71	14.83	17.16	19.80	33.27	116.00	174.59	189.26	141.02	119.90
680	11.94	12.76	13.27	14.94	16.27	20.98	38.26	134.14	189.19	188.11	162.02	136.28
700	10.96	12.07	13.00	13.55	15.47	22.66	50.25	174.01	260.66	246.92	207.92	173.13
720	10.57	11.22	12.03	13.50	16.38	22.01	48.69	158.58	225.67	208.94	184.80	140.94
740	10.72	11.62	12.33	14.10	15.80	21.44	37.08	137.10	173.46	196.04	177.51	135.26
760	11.15	11.78	12.57	12.98	15.09	19.01	37.48	119.31	205.66	197.80	159.23	141.74
780	11.49	12.59	13.13	14.25	16.47	21.96	37.88	141.00	197.05	216.20	199.93	129.97

Table H.6: Data for theoretical latency

Seed	Traffic level											
	5	7	10	14	19	27	37	52	73	102	143	200
200	11.33	12.76	12.79	13.98	16.21	19.65	33.93	100.43	177.47	162.64	162.42	121.91
230	11.82	12.07	13.02	14.30	16.04	20.95	37.99	129.92	197.59	199.23	166.00	134.24
260	10.74	12.24	12.71	13.62	16.11	21.21	43.34	120.83	177.72	191.49	180.24	134.63
290	11.69	12.25	12.35	13.76	15.53	20.20	33.16	125.83	197.36	207.70	179.92	135.81
320	10.41	11.55	12.48	14.33	15.47	19.42	32.05	106.23	166.00	158.33	152.98	116.16
360	10.25	11.24	11.71	12.73	15.39	20.37	44.38	142.88	180.91	192.65	184.33	125.91
380	11.88	12.08	12.75	14.03	16.01	19.43	33.94	128.53	205.65	207.34	160.56	135.79
410	12.58	12.58	12.84	15.27	17.26	21.84	36.60	116.08	178.94	181.10	159.71	129.82
460	11.31	11.82	13.20	14.11	15.88	20.60	40.57	152.78	205.06	197.19	179.49	150.91
470	11.54	11.64	12.86	14.29	16.69	20.12	37.16	121.51	181.87	194.37	147.26	132.92
500	12.56	12.70	14.09	13.95	16.32	20.30	35.83	110.22	170.01	181.12	164.81	116.32
540	13.20	13.74	14.85	16.81	18.39	24.20	41.57	132.08	187.67	189.29	186.01	135.37
560	10.78	11.83	12.34	13.55	16.12	20.23	39.07	138.25	208.19	218.11	185.76	128.75
580	11.01	11.84	12.52	13.71	17.17	19.38	32.63	104.23	165.37	167.77	151.17	116.76
600	10.81	12.00	12.75	13.85	16.53	20.56	37.15	124.21	188.69	213.92	182.27	131.27
620	11.42	12.09	12.49	14.21	15.89	21.24	31.31	93.69	157.22	175.16	147.97	103.90
640	12.69	12.96	13.74	14.95	17.76	20.58	34.45	128.53	189.56	200.21	185.11	117.70
660	12.11	12.89	13.13	14.66	17.19	21.54	36.44	142.58	219.67	208.47	194.06	140.52
680	12.12	13.14	13.86	15.07	16.89	22.01	36.50	119.74	190.72	195.49	163.35	142.54
700	11.54	12.64	13.66	14.59	15.96	21.66	37.14	131.57	201.91	202.62	184.65	127.09
720	10.59	12.58	13.11	14.27	16.82	19.54	39.13	119.28	189.01	184.56	165.66	115.48
740	11.10	12.26	12.53	13.53	14.59	19.48	30.54	93.84	152.92	155.40	155.42	91.34
760	13.85	15.17	16.22	15.83	17.56	21.43	30.68	86.88	156.33	169.56	147.87	116.42
780	10.74	11.99	12.61	13.62	16.58	21.62	37.94	126.65	181.94	219.03	169.75	124.42

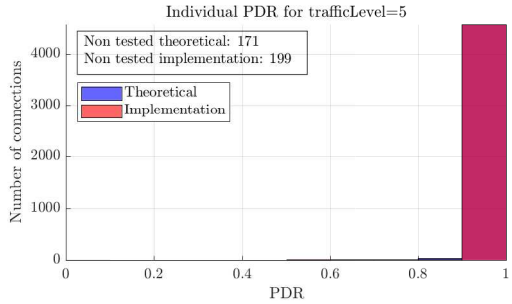
Table H.7: Data for implementation average hops

seed	traffic level											
	5	7	10	14	19	27	37	52	73	102	142	200
200	1.84	1.87	1.90	1.85	1.90	1.86	1.89	1.86	1.80	1.63	1.59	1.44
230	1.73	1.72	1.74	1.74	1.72	1.73	1.69	1.67	1.63	1.52	1.49	1.35
260	1.78	1.83	1.83	1.80	1.86	1.78	1.84	1.75	1.75	1.64	1.64	1.47
290	1.80	1.88	1.85	1.84	1.81	1.82	1.80	1.81	1.77	1.64	1.62	1.49
320	1.75	1.74	1.71	1.75	1.75	1.78	1.70	1.68	1.63	1.54	1.45	1.35
360	1.79	1.89	1.87	1.86	1.85	1.86	1.91	1.83	1.74	1.73	1.56	1.41
380	1.79	1.82	1.80	1.77	1.75	1.77	1.78	1.78	1.70	1.62	1.55	1.40
410	1.86	1.89	1.84	1.93	1.85	1.88	1.85	1.87	1.84	1.65	1.57	1.34
460	1.75	1.72	1.74	1.74	1.74	1.69	1.73	1.69	1.63	1.56	1.50	1.44
470	1.78	1.76	1.80	1.81	1.80	1.80	1.86	1.80	1.75	1.62	1.53	1.41
500	1.79	1.73	1.73	1.73	1.72	1.74	1.76	1.68	1.69	1.61	1.51	1.36
540	1.87	1.83	1.86	1.82	1.80	1.83	1.86	1.86	1.76	1.65	1.54	1.43
560	1.82	1.77	1.83	1.78	1.83	1.81	1.80	1.81	1.70	1.67	1.58	1.44
580	1.79	1.72	1.81	1.80	1.82	1.82	1.82	1.74	1.73	1.67	1.57	1.41
600	1.79	1.83	1.77	1.80	1.82	1.78	1.83	1.75	1.66	1.57	1.56	1.43
620	1.72	1.72	1.76	1.76	1.72	1.78	1.74	1.69	1.69	1.58	1.46	1.38
640	1.94	1.90	1.86	1.95	1.98	1.94	2.02	2.06	1.81	1.69	1.66	1.46
660	1.78	1.80	1.75	1.82	1.86	1.78	1.79	1.79	1.77	1.63	1.44	1.38
680	1.85	1.83	1.84	1.84	1.83	1.85	1.85	1.80	1.71	1.63	1.53	1.42
700	1.80	1.90	1.89	1.82	1.83	1.85	1.86	1.82	1.81	1.73	1.63	1.52
720	1.78	1.78	1.78	1.80	1.80	1.77	1.82	1.79	1.73	1.66	1.56	1.42
740	1.73	1.75	1.75	1.79	1.78	1.77	1.75	1.73	1.62	1.61	1.53	1.40
760	1.78	1.80	1.83	1.78	1.74	1.78	1.76	1.74	1.73	1.62	1.50	1.46
780	1.78	1.78	1.82	1.80	1.79	1.84	1.81	1.83	1.74	1.66	1.60	1.44

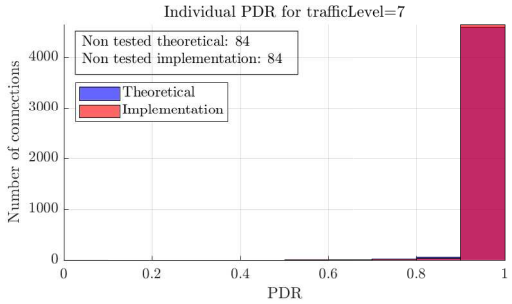
Table H.8: Data for theoretical average hops

Seed	Traffic level											
	5	7	10	14	19	27	37	52	73	102	143	200
200	1.76	1.81	1.77	1.79	1.84	1.76	1.80	1.74	1.67	1.52	1.53	1.39
230	1.80	1.76	1.80	1.81	1.78	1.78	1.78	1.76	1.69	1.64	1.52	1.39
260	1.78	1.86	1.78	1.77	1.78	1.79	1.82	1.74	1.72	1.59	1.54	1.41
290	1.80	1.83	1.85	1.76	1.77	1.78	1.76	1.78	1.70	1.65	1.59	1.42
320	1.72	1.77	1.73	1.80	1.81	1.74	1.74	1.73	1.66	1.52	1.48	1.38
360	1.71	1.78	1.78	1.81	1.77	1.78	1.81	1.76	1.68	1.64	1.59	1.40
380	1.84	1.76	1.77	1.79	1.78	1.76	1.76	1.75	1.73	1.63	1.49	1.41
410	1.77	1.81	1.82	1.83	1.83	1.86	1.82	1.78	1.76	1.59	1.51	1.38
460	1.79	1.79	1.80	1.79	1.82	1.76	1.79	1.82	1.74	1.60	1.52	1.45
470	1.81	1.79	1.79	1.81	1.81	1.80	1.81	1.74	1.68	1.62	1.48	1.41
500	1.79	1.78	1.78	1.73	1.75	1.77	1.83	1.82	1.71	1.64	1.48	1.37
540	1.93	1.86	1.93	1.94	1.92	1.97	1.95	1.91	1.79	1.61	1.59	1.40
560	1.81	1.83	1.79	1.81	1.83	1.79	1.79	1.80	1.78	1.69	1.58	1.41
580	1.76	1.74	1.76	1.79	1.87	1.80	1.80	1.70	1.67	1.58	1.49	1.37
600	1.82	1.86	1.80	1.82	1.87	1.80	1.87	1.77	1.71	1.67	1.57	1.41
620	1.74	1.77	1.71	1.76	1.76	1.82	1.75	1.70	1.64	1.56	1.48	1.34
640	1.85	1.85	1.83	1.86	1.88	1.85	1.88	1.90	1.75	1.65	1.60	1.38
660	1.82	1.85	1.80	1.84	1.87	1.78	1.84	1.81	1.80	1.63	1.57	1.40
680	1.83	1.84	1.85	1.85	1.82	1.84	1.87	1.82	1.75	1.63	1.51	1.45
700	1.82	1.81	1.81	1.84	1.78	1.85	1.79	1.79	1.74	1.62	1.60	1.40
720	1.77	1.78	1.76	1.81	1.81	1.76	1.84	1.78	1.72	1.59	1.51	1.39
740	1.70	1.75	1.71	1.74	1.75	1.76	1.72	1.68	1.61	1.48	1.46	1.31
760	1.88	1.91	1.90	1.87	1.81	1.85	1.85	1.83	1.79	1.58	1.46	1.39
780	1.75	1.79	1.82	1.78	1.78	1.80	1.76	1.76	1.68	1.66	1.53	1.39

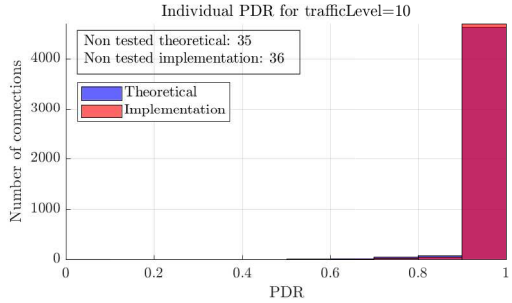
Test data: Individual PDR histograms



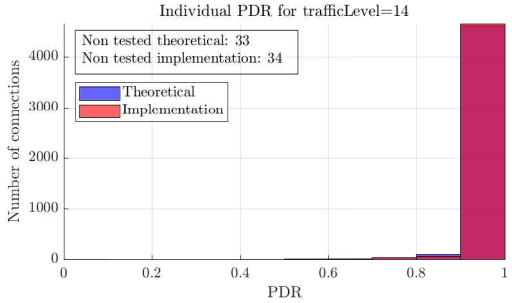
(a) Individual Packet Delivery Ratio (PDR) when the traffic level is 5 packets per second.



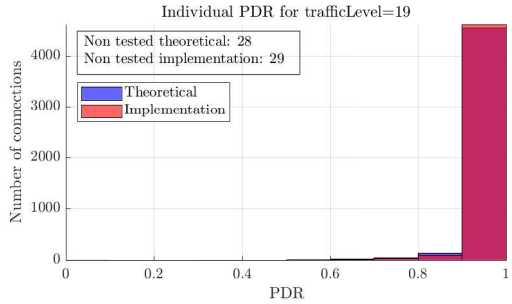
(b) Individual PDR when the traffic level is 7 packets per second.



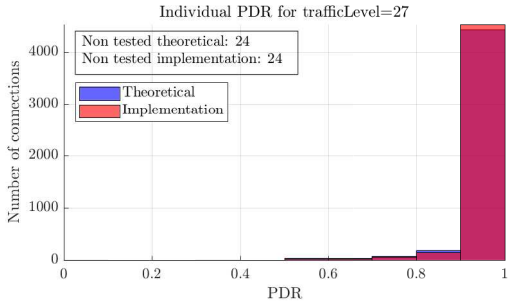
(c) Individual PDR when the traffic level is 10 packets per second.



(d) Individual PDR when the traffic level is 14 packets per second.

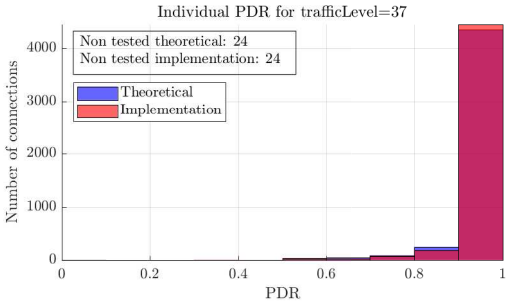


(e) Individual PDR when the traffic level is 19 packets per second.

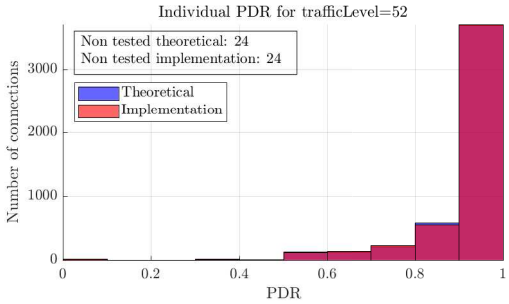


(f) Individual PDR when the traffic level is 27 packets per second.

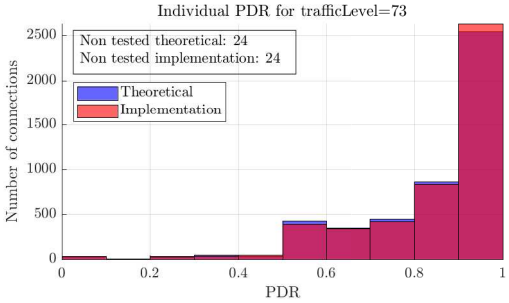
Figure I.1: Individual PDR when the traffic level is between 5 and 27 packets per second.



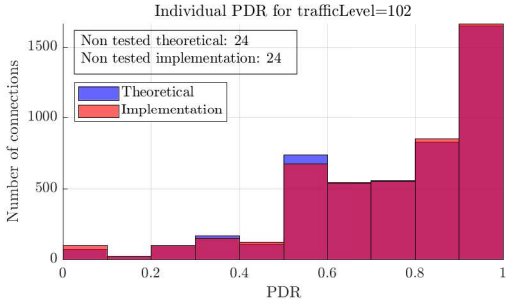
(a) Individual PDR when the traffic level is 37 packets per second.



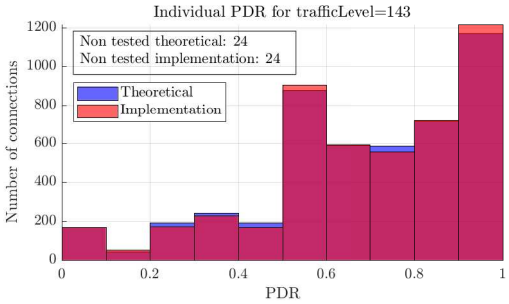
(b) Individual PDR when the traffic level is 52 packets per second.



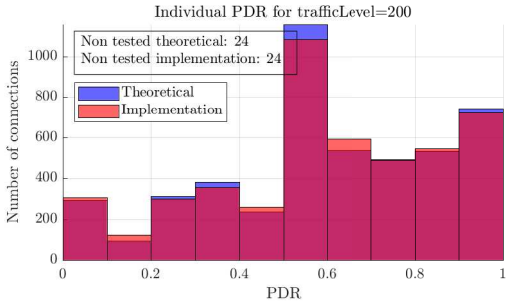
(c) Individual PDR when the traffic level is 73 packets per second.



(d) Individual PDR when the traffic level is 102 packets per second.



(e) Individual PDR when the traffic level is 143 packets per second.



(f) Individual PDR when the traffic level is 200 packets per second.

Figure I.2: Individual PDR when the traffic level is 37 to 200 packets per second.

Code implementation: Highest connectivity based greedy algorithm J

Code Snippet J.1: Code implementation - Highest connectivity based greedy algorithm.

```
function relay = connectedDominatingSet( ~,options,BlockER,PL )
%CONNECTEDDOMINATINGSET Creates relay destribution
%  [ relay ] = connectedDominatingSet( nodes,options,BER,PL )

limit = BLEP2Limit(BlockER,options);
% Calculate reachables
% Find reachable nodes for simple simmulator:
reachables = (PL >= limit) .* (~eye(options.amountNodes));
relay = zeros(options.amountNodes,1); % List of who are relays
dominated = zeros(options.amountNodes,1); % List of who are dominated
clusterID = (1:options.amountNodes)';
setupStep = [zeros(options.amountNodes,1),(1:options.amountNodes)'];


while ( sum(setupStep(:,1) ~= -1 ) )

    % Choose a node in step 0, if all are done with step 0, choose one in
    % step 1.
    if ( any (setupStep(:,1) == 0) )
        % Extract the nodes who are in step 0.
        extractedNodes = setupStep(:,1) == 0;
        temp = setupStep(extractedNodes,:);

        % Choose a node to continue with the next step.
        [n,~] = size(temp);
        node = randomtime(1,n);
        nodeID = temp( node,2 );
        nodeStep = temp( node,1 );
    else
        % Extract the nodes who are in step 1.
        extractedNodes = setupStep(:,1) == 1;
        temp = setupStep(extractedNodes,:);

        % Choose a node to continue with the next step.
        [n,~] = size(temp);
        node = randomtime(1,n);
        nodeID = temp( node,2 );
        nodeStep = temp( node,1 );
    end
```

```
40      % Do the next step.
41      switch nodeStep
42          case 0 % Check neighboursAmount & neighboursNeighboursAmount
43              I = find( reachables(:,nodeID) == 1 ); % Find neighbours
44              if numel(I) == 0
45                  error('Nodes without neighbours')
46                  setupStep(nodeID,1) = -1;
47              end
48
49              % Remove relays and dominated:
50              for n = find(relay)'
51                  temp = find ( I == n );
52                  I(temp) = []; %delete the elements who are relays
53              end
54
55              IDominatedRemoved = I;
56              for n = find(dominated)'
57                  temp = find ( IDominatedRemoved == n );
58                  IDominatedRemoved(temp) = []; %delete the elements who are dominated
59              end
60              neighboursAmount = numel(IDominatedRemoved);
61              neighboursNeighboursAmount = [];
62              for temp = I'
63                  neighboursNeighbours = find( reachables(:,temp) == 1);
64                  % Remove relays and dominated:
65                  for n = find(relay)'
66                      temp2 = find ( neighboursNeighbours == n );
67                      neighboursNeighbours(temp2) = []; %delete the elements who are relays
68                  end
69                  for n = find(dominated)'
70                      temp2 = find ( neighboursNeighbours == n );
71                      neighboursNeighbours(temp2) = []; %delete the elements who are dominated
72                  end
73                  neighboursNeighboursAmount(end+1) = numel(neighboursNeighbours);
74              end
75
76              [m,entry]= max(neighboursNeighboursAmount);
77              if (neighboursAmount >= m) % I am dominator
78                  dominatorID = nodeID; % The ID of the node
79                  relay(dominatorID) = 1;
80
81                  % Before we head on, we remove the entries from I who are
82                  % relays to avoid errors.
83                  for n = find(relay)'
84                      temp = find ( I == n );
85                      I(temp) = []; %delete the elements who are relays
86                  end
87              else % Neighbour is dominator
88                  dominatorID = I(entry); % The ID of the dominating neighbour
89                  relay(dominatorID) = 1;
90
91                  % Make all dominating neighbour neighbours dominated:
92                  % Create list of neighboursNeighbours
93                  I = find( reachables(:,dominatorID) == 1 ); % Find neighbours
94
95                  % Before we head on, we remove the entries from I who are
96                  % relays to avoid errors.
```

```
97                for n = find(relay)'
98                    temp = find ( I == n );
99                    I(temp) = []; %delete the elements who are relays
100               end
101
102               % Dirty fix if PL is not being symmetric.
103               dominated(nodeID) = 1;
104               setupStep(nodeID,1) = 1;
105               clusterID(nodeID) = -1;
106           end
107
108           % Assign roles and clusterIDs:
109           dominated(I) = 1;
110           dominated(dominatorID) = 0;
111           clusterID(I) = -1;
112           clusterID(dominatorID) = dominatorID;
113           setupStep(I,1) = 1;
114           setupStep(dominatorID,1) = -1;
115
116           % Allign clusterIDs
117           clusterID =
                  allignClusterIDs(dominatorID,reachables,relay,dominated,clusterID,setupStep);
118
119       case 1 % Connect relays
120           % Check which neighbours are relays, and if they can see each
121           % other.
122           I = find( reachables(:,nodeID) == 1 ); % Find neighbous
123           % remove nodes who aren't relays.
124           for n = find(~relay)'
125               temp = find ( I == n );
126               I(temp) = []; %delete the elements who aren't relays
127           end
128
129           seenCIDs = clusterID(I);
130
131           if numel(unique(seenCIDs)) == 1 % See what relays neighbours can see
132
133               I = find( reachables(:,nodeID) == 1 );
134               % Remove nodes who are relays.
135               for n = find(relay)'
136                   temp = find ( I == n );
137                   I(temp) = []; %delete the elements who are relays
138               end
139
140               % Check visible CIDs of neighbours
141               for k = I' %Must be in rows and not a vector
142                   Ineighbour = find( reachables(:,k) == 1 ); % Find neighbous
143                   % remove nodes who aren't relays.
144                   for n = find(~relay)'
145                       temp = find ( Ineighbour == n );
146                       Ineighbour(temp) = []; %delete the elements who aren't relays
147                   end
148                   % Check if any neighbours can see a CID that I can not
149                   CIDCheck1 = ismember( seenCIDs,clusterID(Ineighbour) );
150                   CIDCheck2 = ismember( clusterID(Ineighbour),seenCIDs );
151                   CIDCheck = sum(~CIDCheck1) | sum(~CIDCheck2);
152
```

```
153                     if CIDCheck %Is true if one of the two nodes can see different CIDs
154                         relay(nodeID) = 1;
155                         relay(k) = 1;
156                         dominated(nodeID) = 0;
157                         dominated(k) = 0;
158                         clusterID(nodeID) = nodeID;
159                         clusterID(k)=k;
160 
161                         % Allign clusterIDs
162                         clusterID =
                                allignClusterIDs(nodeID,reachables,relay,dominated,clusterID,setupStep);
163                         clusterID =
                                allignClusterIDs(k,reachables,relay,dominated,clusterID,setupStep);
164 
165                         % Set node k as finished
166                         setupStep(k,1) = -1;
167                         break;
168                     end
169                 end
170 
171             elseif numel(unique(seenCIDs)) > 1 % Connect the relays.
172                 relay(nodeID) = 1;
173                 dominated(nodeID) = 0;
174                 clusterID(nodeID) = nodeID;
175 
176                 % Allign clusterIDs
177                 clusterID =
                        allignClusterIDs(nodeID,reachables,relay,dominated,clusterID,setupStep);
178             else
179                 disp('Error␣in␣connectedDominatingSet␣case␣1')
180             end
181 
182             setupStep(nodeID,1) = -1; % Node has completed setup.
183 
184         otherwise
185             disp('Error␣in␣connectedDominatingSet.m')
186     end
187 end
188 
189 relay = find(relay)';
190 
191 end
```

Code implementation: Central highest connectivity based greedy algorithm

K

Code Snippet K.1: Code implementation - Central Highest connectivity based greedy algorithm.

```
function [rel_lst] = BermanAlgo( nodes,options,BlockER,PL )
BLEPLIMIT = options.LOSLimit;
BER = 1 - nthroot((1-BLEPLIMIT),nodes{1}.options.messageLength*8);
SINR = (erfcinv(BER*2)^2)/0.68;
% Derived from: BER = gather(0.5* erfc(sqrt(0.68*SINR))); %AWGN
sensitivity = 10^(nodes{1}.options.receiverSensitivity/10);
limit = SINR*sensitivity;
PL_draw = PL;
BLEP = 1-(1-0.5.*erfc(sqrt(0.68.*PL./(sensitivity)))).^(47*8);
relay = zeros(options.amountNodes,1);

reachables = (PL >= limit) .* (~eye(options.amountNodes));
clusterID = zeros(options.amountNodes,1);
dominated = 0;
setupStep = 0;
% Check if map is fully connected
CheckConn = sum(reachables);
b = (find(CheckConn == 0 ));
if b
    error('Could not connect')
    disp('ERROR map not properly connected cant find relays')
    disp('No relay list can be found ')
    relay = ones(options.amountNodes,1);
    disp('All relays has been made because no set could be found!')
    rel_lst = find(relay)';
    return
end

%Phase 1 find the biggest
[~, biggest] = max(sum(reachables));
I = (~reachables(:,biggest)*ones(1,numel(nodes))).*reachables;
rel_lst = biggest;
relay(rel_lst) = 1;
clusterID(biggest) = biggest;
clusterID = allignClusterIDs(biggest,reachables,relay,dominated,clusterID,setupStep);
```

```
for n=1:(options.amountNodes-1)
    [~, biggest] = max(sum(I));
    I = (~I(:,biggest)*ones(1,numel(nodes))).*I; %removing relays and grey nodes from I
        to avoid errors
    rel_lst(end+1) = biggest;
    relay(rel_lst) = 1;
    clusterID(biggest) = biggest;
    clusterID = allignClusterIDs(biggest,reachables,relay,setupStep,clusterID,setupStep);
        %makes sure they are connected through allignment of ID's
    if sum(sum(I)) == 0
        break
    end
end
countID1 = numel(unique(clusterID(clusterID~=0)));
%Phase2 connection
run = 0;
dispel = 0;
count = 0;
while numel(unique(clusterID(clusterID~=0)))~= 1
    count = count + 1;
    if count > 10
        error('could probably not connect this set')
    end

    for n = 1:numel(rel_lst)
        nodeID = rel_lst(1,n);
        I = find( reachables(:,nodeID) == 1 ); % Find neighbous
        % remove nodes who aren't relays.
        for n = find(~relay)'
            temp = find ( I == n );
            I(temp) = []; %delete the elements who aren't relays
        end

        seenCIDs = clusterID(I);

        if numel(unique(seenCIDs)) == 1 % See what relays neighbours can see

            I = find( reachables(:,nodeID) == 1 );
            % Remove nodes who are relays.
            for n = find(relay)'
                temp = find ( I == n );
                I(temp) = []; %delete the elements who are relays
            end

            % Check visible CIDs of neighbours
            for k = I' %Must be in rows and not a vector
                Ineighbour = find( reachables(:,k) == 1 ); % Find neighbous
                % remove nodes who aren't relays.
                for n = find(~relay)'
                    temp = find ( Ineighbour == n );
                    Ineighbour(temp) = []; %delete the elements who aren't relays
                end
                % Check if any neighbours can see a CID that I can not
                CIDCheck1 = ismember( seenCIDs,clusterID(Ineighbour) );
                CIDCheck2 = ismember( clusterID(Ineighbour),seenCIDs );
                CIDCheck = sum(~CIDCheck1) | sum(~CIDCheck2);

```

```
91                  if CIDCheck %Is true if one of the two nodes can see different CIDs
92                      relay(nodeID) = 1;
93                      relay(k) = 1;
94                      dominated(nodeID) = 0;
95                      dominated(k) = 0;
96                      clusterID(nodeID) = nodeID;
97                      clusterID(k)=k;
98
99                      % Allign clusterIDs
100                     clusterID =
                            allignClusterIDs(nodeID,reachables,relay,dominated,clusterID,setupStep);
101                     clusterID =
                            allignClusterIDs(k,reachables,relay,dominated,clusterID,setupStep);
102                     cnt = 1;
103                     %used for checking only 2 clusters are formed.
104                     countID = numel(unique(clusterID(clusterID~=0)));
105                     %used for debugging
106                     break;
107
108                 end
109                 run = run +1; % checks if the countID hasn't been done
110                 if run == options.amountNodes %check all nodes... no neighbours found
111                     dispel = 1;
112                     %disp('clusters found')
113                     break; %need to find a 2 neighbour connection.
114                 end
115             end% dirty fix to get out of stuck loop
116
117
118             % Only in the case that the previous code did not connect
119             % properly now checking for 2 pairs.
120             if dispel == 1
121                 %if 2 unique cluster ID has can be seen, connect them through 2 nodes.
122
123                 %find reachable from the uniqueID's
124                 %see if any of its neighbours will connect, if yes
125                 %connect them, if no check both ids with rest of
126                 %rel_lst neighbours
127                 x = unique(clusterID);
128                 uni = x(2:end);
129                 uniq = uni;
130                 %this is for all other IDS
131                 % Checks each unique cluster ID if it has neighbours
132                 % that can see another cluster ID and picks that one if
133                 % it exist
134                 for w = 1:(numel(uniq))-1
135                     mem = uni(w);
136                     member = find( reachables(:,mem) == 1 );
137                     for y = 1:numel(member)
138                         I_new = find( reachables(:,member(y)) == 1 );% find reachables of
                                the nodes of members
139                         for z = 1:numel(I_new)
140                             I_new2 = find(reachables(:,I_new(z)) == 1 ); % finds reachables
                                    of the neighbouring node of members
141                             %Checking nightmare: %needs to check if member and if
142                             %relay plus if they are the same discard.
143                             CheckPairAll1 = ismember(I_new,I_new2);
```

```
144
145                         CheckPairAll2 = ismember(rel_lst,I_new2);
146
147                         relaylistID = rel_lst(CheckPairAll2);
148
149                         CheckID = clusterID(relaylistID);
150                         MemberID = clusterID(member);
151                         sumCheckID = sum(CheckID)/numel(find(CheckID));
152                         sumCheckMem = sum(MemberID)/numel(find(MemberID));
153                         %if statement here if check pair sum is true
154                         if sumCheckID > uni(w) & CheckID ~= sumCheckMem
155                             %there is a issue here
156                             %each time i goes into this loop it does not connect
157                             %set element in CheckPairAll1 that is 1 to be relay
158                             %and set y to be relay aswell
159                             k = I_new(z);
160                             km = member(y);
161                             relay(km) = 1;
162                             relay(k) = 1;
163                             dominated(km) = 0;
164                             dominated(k) = 0;
165                             clusterID(km) = km;
166                             clusterID(k)=k;
167                             clusterID =
                                    allignClusterIDs(km,reachables,relay,dominated,clusterID,setupStep);
168                             clusterID =
                                    allignClusterIDs(k,reachables,relay,dominated,clusterID,setupStep);
169                             %Used for debugging to figure out if ID have been
170                             %aligned visually
171                             countID = numel(unique(clusterID(clusterID~=0)));
172
173                             x = unique(clusterID);
174                             uniq = x(2:end);
175
176                             counter = numel(find(relay));
177                             break;
178                         end
179                     end
180                 end
181             end
182         end
183     end
184 end
185
186
187     rel_lst = find(relay)';
188 end
189 end
```

Code implementation: Two hop pruning algorithm

L

Code Snippet L.1: Code implementation - K-2 Pruning algorithm.

```
function [rel_lst] = K2PruningRelayAlgorithm(nodes,options,BlockER,PL)

%% Calculation the limit of needed signal strengt to have a connection
Limit = BLEP2Limit(BlockER,options);

%% Remove all existing relays
rel_lst = zeros(1,options.amountNodes); % zero vector
for n = 1:(options.amountNodes)
    nodes{n}.reachable = [];
end

%% finding connections
for n = 1:options.amountNodes % check once for all nodes
    for m = 1:options.amountNodes % check against all other nodes
        if (m ~= n) % Only if it is not looking at itself
            if (PL(n,m) >= Limit) % if signal is abouve a certain limit node n can see
                node m
                nodes{n}.reachable(end+1)= m; %add node m's number to list of node n's
                    reachables
            end
        end
    end
end

%% The construction phase
%disp(['Start: ' num2str(sum(rel_lst))])
for n = 1:(options.amountNodes) % check for all nodes
    for m = nodes{n}.reachable % check for all neighbours of n
        include = sum( ismember(nodes{n}.reachable, nodes{m}.reachable)); % counts n's
            neighbour which is also in m's neighbour list
        if (include ~= numel(nodes{n}.reachable) - 1) % if m do not cover all n's
            neighbours. -1 because n will be in m's neighbours list
            rel_lst(n) = 1; % n is added to the relay list
            break
        end
    end
end

%% The fisrt reduction phase
%disp(['Before step 1: ' num2str(sum(rel_lst))])
for n = find(rel_lst) % check for all relay nodes
    for m = nodes{n}.reachable % check for all neighbours of n
        if (rel_lst(m) == 1) % if m is relay
```

```
40              include = sum( ismember(nodes{n}.reachable, nodes{m}.reachable)); % counts n's
                    neighbour which is also in m's neighbour list
41              if ( include == numel(nodes{n}.reachable) - 1) % if m covers all n's
                    neighbours. -1 because n will be in m's neighbours list
42                  rel_lst(n) = 0;
43                  break
44              end
45          end
46      end
47  end
48
49  %% The second reduction phase
50  %disp(['Before step 2: ' num2str(sum(rel_lst))])
51  for n = find(rel_lst) % check for all relay nodes
52      if (rel_lst(n) == 1)
53          nRelayNeighbours = nodes{n}.reachable;
54          nRelayNeighbours = nRelayNeighbours( ismember(nodes{n}.reachable,find(rel_lst)));
55          for m = nRelayNeighbours % Check for all neighbours of n
56              if (rel_lst(m)==1)
57                  nmRelayNeighbours =
                        nRelayNeighbours(ismember(nRelayNeighbours,nodes{m}.reachable));
58                  for k = nmRelayNeighbours % check for all relay nodes
59                      if (k ~= n && k~=m) && (rel_lst(k) == 1)
60                          %pair = unique([nodes{n}.reachable' ; nodes{m}.reachable']);
61                          pair = [nodes{n}.reachable, nodes{m}.reachable]; % which nodes do n
                                and m span combined
62                          inPair = sum(ismember(nodes{k}.reachable, pair)); % counts who many
                                of k's neighbours are covered by pair
63                          if ( inPair == numel(nodes{k}.reachable))
64                              rel_lst(k) = 0; % Fjern k fra relay listen
65                          end
66                      end
67                  end
68              end
69          end
70      end
71  end
72
73  %% The third reduction phase
74  %disp(['Before step 3: ' num2str(sum(rel_lst))])
75  for n = find(rel_lst) % check for all relay nodes
76      m = nodes{n}.reachable;
77      m = m( ismember(m,find(rel_lst)));
78      for k = m
79          nmRelayNeighbours = m(ismember(m,nodes{k}.reachable));
80          mNeighbours = [];
81          for temp=[k,nmRelayNeighbours]
82              mNeighbours=[mNeighbours,nodes{temp}.reachable];
83          end
84          mNeighbours = unique(mNeighbours);
85          count = sum(ismember(nodes{n}.reachable, mNeighbours)); % counts who many of k's
                neighbours are covered by pair
86          if ( count == numel(nodes{n}.reachable))
87              rel_lst(n) = 0; % Fjern k fra relay listen
88          end
89      end
90  end
```

```
91  %disp(['After step 3: ' num2str(sum(rel_lst))])
92
93  %% Convert output
94  rel_lst = find(rel_lst); % convert relay list from [1 0 1 1 0] => [1 3 4]
95
96  %% Error detection
97  %disp(['Error detection started ']);
98  for n = 1:options.amountNodes % check once for all nodes
99      if (sum( ismember(nodes{n}.reachable, rel_lst)) == 0)
100         disp(['Error!!! Node: ' num2str(n) ' cant see a relay']);
101     end
102 end
103
104 for n = rel_lst % check for all relay nodes
105     if (sum( ismember(nodes{n}.reachable, rel_lst))==0)
106         disp(['Error!!! Relay node: ' num2str(n) ' cant see a another relay']);
107     end
108 end
109
110 end
```

Code implementation: First come algorithm

M

Code Snippet M.1: Code implementation - First come algorithm.

```
function [rel_lst] = IamTheDominatorBaby(nodes,options,BlockER,PL)

%% Calculation the limit of needed signal strengt to have a connection
limit = BLEP2Limit(BlockER,options);

%% find neighbour nodes
reachables = (PL >= limit) .* (~eye(options.amountNodes));

%% The construction
NodeVector = 1:(options.amountNodes); % make vector: [1 2 3 ... "options.amountNodes"]
RandomNodeVector = NodeVector(randperm(length(NodeVector))); % randomly shuffel the
    vector entries

dominated = zeros(options.amountNodes,1);
rel_lst1 = zeros(1,options.amountNodes);

for n = RandomNodeVector
    if (dominated(n) == 0)
        rel_lst1(n) = 1;
        dominated = dominated + reachables(:,n) * eye;
    end
end

%% The first connection phase
C = [0; 0];
rel_lst2 = zeros(1,options.amountNodes);

for u = RandomNodeVector % check for all nodes
    if ( dominated(u) > 0 && rel_lst1(u) == 0 )
        if sum(ismember(find(reachables(u,:)), find(rel_lst1))) >= 2 % can u see 2 relays?
            D1 = find(ismember(find(reachables(u,:)), find(rel_lst1))); % Entries of
                reachables(u,:) who are relays
            D2 = find(reachables(u,:)); % IDs of neighbours
            D3 = D2(D1); % The relays u can see

            W = nchoosek(D3,2); % the different ways to combine the found relays. no
                order, no replacement

            for n = 1:numel(W(:,1))
                w1 = W(n,1);
                w2 = W(n,2);
                counter = 0;

```

```
41                for n = 1:numel(C(1,:))
42                    if ( sum(ismember([w1; w2], C(:,n))) == 2 )
43                        break
44                    elseif ( sum(ismember([w1; w2], C(:,n))) < 2 )
45                        counter = counter + 1;
46                        if counter == numel(C(1,:))
47                            C(:,end+1) = [w1; w2];
48                            rel_lst2(u) = 1;
49                                                  %disp(['node: ' num2str(w1) ' og '
                                                        num2str(w2) ' blev forbundet af: '
                                                        num2str(u)]);
50                        end
51                    end
52                end
53            end
54        end
55    end
56 end
57 rel_lst12 = rel_lst1 + rel_lst2;
58
59 % find the new vector of dominated nodes
60 dominated = zeros(options.amountNodes,1);
61 for n = find(rel_lst12)
62     dominated = dominated + ( reachables(:,n) * eye);
63 end
64
65 %% The second connection phase
66 rel_lst3 = zeros(1,options.amountNodes);
67 for u = find(rel_lst1)
68     % vil break her til
69     bf = 0;
70     for v = find(rel_lst1)
71         if u ~= v && sum(ismember(v, find(reachables(u,:)))) == 0
72             % hvis der ikke allerede er et relay imellem dem
73
74             D1 = find(ismember(find(reachables(u,:)), find(rel_lst1))); % hvad plads i
                    reachables(u,:) er de fundne relays
75             D2 = find(reachables(u,:));
76             uNaboRelays = D2(D1); %the relays u can see
77
78             D1 = find(ismember(find(reachables(v,:)), find(rel_lst1)));
79             D2 = find(reachables(v,:));
80             vNaboRelays = D2(D1);
81
82             if ( sum( ismember(uNaboRelays, vNaboRelays) ) == 0 )
83                 counter = 0;
84                 for n = 1:numel(C(1,:))
85
86                     if ( sum(ismember([u; v], C(:,n))) == 2 )
87                         break
88                     elseif ( sum(ismember([u; v], C(:,n))) < 2 )
89                         counter = counter + 1;
90                         if counter == numel(C(1,:))
91                             % u og v er ikke forbundet
92                             n1 = find(reachables(u,:));
93                             for m = n1
94                                 n2 = find(reachables(m,:));
```

```
                            for l = n2
                                if l~=m && sum(ismember(l, find(reachables(v,:)))) == 1
                                    % node l of m som er naboer
                                    % forbinder node u og v
                                    C(:,end+1) = [u; v];
                                    rel_lst3(l) = 1;
                                    rel_lst3(m) = 1;
                                    %disp(['node: ' num2str(u) ' og ' num2str(v) ' blev
                                        forbundet af: ' num2str(l) ' og ' num2str(m)]);
                                    bf = 1; % vil break herfra
                                    break;
                                end
                            end
                            if bf == 1
                                break; % check break flag
                            end
                        end
                    end
                end
                if bf == 1
                    break; % check break flag
                end
            end
        end
        if bf == 1
            break; % check break flag
        end
    end
end
rel_lst13 = rel_lst1 + rel_lst3;

%% Convert output
rel_lst123 = rel_lst1 + rel_lst2 + rel_lst3;
rel_lst = rel_lst123;

rel_lst = find(rel_lst); % convert relay list from [1 0 1 1 0] => [1 3 4]
end
```

Code implementation: Highest connectivity based greedy algorithm implementation

N

In Code snippet N.1 the behaviour of the algorithm when a "Network Setup Packets" is received is seen. In Code snippet N.2 the behaviour of the algorithm when a "Network Setup Packets" is transmitted is seen.

Code Snippet N.1: Code implementation - Highest connectivity based greedy algorithm implementation. Code showing the behaviour when a "Network Setup Packets" is received.

```
1  function obj = readNetworkSetupMessages(obj)
2          % If i am at step 2 and CID's are changing. I will delay my decision
3          if obj.Step == 2 % am i in step 2
4                  if any(obj.NeighbourMatrix(:,1) == obj.rcMsgSRC) % Have i seen the packet
                       sender before
5                      n = find(obj.NeighbourMatrix(:,1) == obj.rcMsgSRC); % Entry of
                           packet sender in the NeighbourMatrix
6                      if obj.message(2) ~= obj.NeighbourMatrix(n,3) % Have his CID changed
7
8                              % Calculate PBD
9                              PDR = min(
                                   obj.NeighbourMatrix(1:obj.numTrustedNeighbours,11)./(obj.NeighbourMatrix(1:obj
                                   ); %What is the lowest PDR in my FilteredNeighbourMatrix
10                             PBD = ceil(- 3/(log(1 - PDR)) ); % syms PBD; eq = 1 -
                                   (1-PDR)^PBD == 0.95; solve(eq,PBD);
11                             PacketsBeforeArrival = max([PBD, 3]);
12
13                             obj.timeToDecision = obj.time +
                                   1/obj.PPS*2*PacketsBeforeArrival*125000 +
                                   2*obj.maxRndWaitChangeChannel*obj.NumDiscoveryPackets +
                                   PacketsBeforeArrival*3*47; % Delay my decision
14
15                     end
16             end
17         end
18
19         % Update information about neighbours
20         % NeighbourMatrix: ID    Message    min(RSSI)  Packet received counter
21         % Message:        UNC    CID        LNS    Role   Step   BeRelay       PPS
               PacketCNT
22         % 1: ID           : Node ID
```

```
23          % 2: UNC          : Number of Unconnected Neighbours
24          % 3: CID          : Highest Cluster ID
25          % 4: LNS          : Lowest Neighbour Step
26          % 5: Role         : Role of Node
27          % 6: Step         : Step of Node
28          % 7: BeRelay      : ID of Node that Sender want to be relay
29          % 8: PPS          : Packets sent per second
30          % 9: PacketCNT    : Packets sent in this step
31          % 10: min(RSSI)   : Not used, Minimum RSSI received
32          % 11: PRC         : Packets Received counter
33          if isempty(obj.NeighbourMatrix) % First entry in the matrics. First packet
                received
34              obj.NeighbourMatrix(1,:) = [obj.rcMsgSRC obj.message
                    min(obj.RSSI(1:obj.msgLenReceive)) 1];
35          else
36              n = find(obj.NeighbourMatrix(:,1) == obj.rcMsgSRC); % Find entry of packet
                    sender in the NeighbourMatrix
37              if isempty(n) % First packet from this specific sender
38                  obj.NeighbourMatrix(end+1,:) = [obj.rcMsgSRC obj.message
                        min(obj.RSSI(1:obj.msgLenReceive)) 1];
39              else % Update information in NeighbourMatrix
40                  obj.NeighbourMatrix(n,:) = [obj.rcMsgSRC obj.message
                        min([min(obj.RSSI(1:obj.msgLenReceive))
                        obj.NeighbourMatrix(n,6)]) obj.NeighbourMatrix(n,11)+1];
41              end
42          end
43
44          if obj.Step == 0 % Discovery step
45              obj.numTrustedNeighbours = numel(obj.NeighbourMatrix(:,1)); % Trust
                    everyone
46
47              if any(obj.NeighbourMatrix(:,6) > 0) % Someone have advanced to step 1
48                  obj.Step = 1;
49
50                  % Filter my neighbours. Move trusted neighbours to the top of the
                        matrix
51                  obj.TrustedNeighbours = find(obj.NeighbourMatrix(:,11) ./
                        obj.NeighbourMatrix(:,9) >= obj.filteringFactor ... %
                        Received/transmitted >= factor
52                      & obj.NeighbourMatrix(:,11) > obj.MinNumPackets); % Received
                            > min value
53                  notTrusted = find(obj.NeighbourMatrix(:,11) ./
                        obj.NeighbourMatrix(:,9) < obj.filteringFactor |
                        obj.NeighbourMatrix(:,11) <= obj.MinNumPackets ); % The
                        opposite of the trusted
54                  obj.NeighbourMatrix = [
                        obj.NeighbourMatrix(obj.TrustedNeighbours,:);
                        obj.NeighbourMatrix(notTrusted,:)]; % Sort matrix
55                  obj.numTrustedNeighbours = numel(obj.TrustedNeighbours); % Set
                        number of trusted neighbours
56
57                  if obj.numTrustedNeighbours == 0 % If a node do not trust anyone.
                        Pick the best node as trusted
58                      bestFriend = find ( (obj.NeighbourMatrix(:,11) ==
                            max(obj.NeighbourMatrix(:,11))),1 ); % The neighbour
                            with most received packet count is picked
59                      notTrusted = obj.NeighbourMatrix(:,1) ~=
```

```
                                obj.NeighbourMatrix(bestFriend,1);
60                          obj.NeighbourMatrix = [ obj.NeighbourMatrix(bestFriend,:);
                                obj.NeighbourMatrix(notTrusted,:)]; % Sort matrix
61                          obj.numTrustedNeighbours = 1; % Set number of trusted
                                neighbour
62                  end
63
64                  % Adjust PPS according to the amount of nodes in the network
65                  Neighbours = obj.options.amountNodes; %numel(
                        obj.NeighbourMatrix(:,1) );
66                  if Neighbours > 71 % Functio will be bigger then 10 for < 71
67                          a = 11.3505; % Found from curvefit apendix H
68                          b = -0.0017334;
69                          obj.PPS = a*exp(b*Neighbours);
70
71                          if obj.PPS < 0.1
72                                  obj.PPS = 0.1;
73                          elseif obj.PPS > 10
74                                  obj.PPS = 10;
75                          end
76
77                  else
78                          obj.PPS = 10;
79                  end
80                  obj.PacketCNT = 1;
81                  obj.NeighbourMatrix(:,11) = 0; % All entries are set to zero
82          end
83      end
84
85      % If someone have BeRelay set. Add their ID to the waitList
86      temp = find( obj.NeighbourMatrix(1:obj.numTrustedNeighbours,7) ~= 0 );
87      obj.waitList = obj.NeighbourMatrix(temp,1);
88
89  end
```

Code Snippet N.2: Code implementation - Highest connectivity based greedy algorithm implementation. Code showing the behaviour when a "Network Setup Packets" is Transmitted.

```
1   function obj = networkLayerIdleBehaviour(obj)
2       if  isempty(obj.NeighbourMatrix) % First packet send
3
4           % UNC  CID    LNS    Role   Step   BeRelay       PPS    PacketCNT
5           obj.message = [obj.UNC obj.CID obj.LNS obj.Role obj.Step obj.BeRelay
                obj.PPS obj.PacketCNT]; % Generate a message
6
7           % Usage: createMsg(obj,ID,SRC,DST,TTL,len,message)
8           obj = createMsg(obj, obj.PacketID, obj.id, -1, 0, obj.SetupPacketLength,
                obj.message);
9           obj.PacketID = obj.PacketID + 1;
10          obj.PacketCNT = obj.PacketCNT + 1;
11
12          obj.idleDelay = randomtime(1, round((125000)/(obj.PPS/2))); % Schedule
                when to send next packet
13          return % No need to check if we should make any decisions since we just
                send our first packet.
14      end
```

```
15
16          if obj.Step > 0
17                  if any(obj.waitList) ... % Any nodes on the waitList. The waitList
                        contains all nodes with a marked BeRelay
18                                  | any(obj.NeighbourMatrix(1:obj.numTrustedNeighbours,6) <
                                        obj.Step) ... % Any node at a lower step
19                                  | any((obj.NeighbourMatrix(1:obj.numTrustedNeighbours,1) <
                                        obj.id) ... % Any nodes at the same step with a lower id
20                                  & (obj.NeighbourMatrix(1:obj.numTrustedNeighbours,6) ==
                                        obj.Step) ) ...
21                                  | any(obj.NeighbourMatrix(1:obj.numTrustedNeighbours,4) <
                                        obj.Step) % Any nodes see someone at a lower step
22                          if (obj.Step < 3)
23
24                                  % Calculate PBD
25                                  PDR = min(
                                        obj.NeighbourMatrix(1:obj.numTrustedNeighbours,11)./(obj.NeighbourMatrix
                                        ); %What is the lowest PDR in my FilteredNeighbourMatrix
26                                  PBD = ceil(- 3/(log(1 - PDR)) ); % syms PBD; eq = 1 -
                                        (1-PDR)^PBD == 0.95; solve(eq,PBD);
27                                  PacketsBeforeArrival = max([PBD, 3]);
28
29                                  obj.timeToDecision = obj.time +
                                        1/obj.PPS*2*PacketsBeforeArrival*125000 +
                                        2*obj.maxRndWaitChangeChannel*obj.NumDiscoveryPackets +
                                        PacketsBeforeArrival*3*47; % Delay my decision
30                          end
31                  end
32          end
33
34          if obj.timeToDecision <= obj.time % Should i make a decision
35                  switch obj.Step % What is my step
36                          case 0 % I am in the discovery step
37                                  obj.Step = 1;
38
39                                  % Filter my neighbours. Move trusted neighbours to the top
                                        of the matrix
40                                  obj.TrustedNeighbours = find(obj.NeighbourMatrix(:,11) ./
                                        obj.NeighbourMatrix(:,9) >= obj.filteringFactor ... %
                                        Received/transmitted >= factor
41                                     & obj.NeighbourMatrix(:,11) > obj.MinNumPackets); %
                                           Received > min value
42                                  notTrusted = find(obj.NeighbourMatrix(:,11) ./
                                        obj.NeighbourMatrix(:,9) < obj.filteringFactor |
                                        obj.NeighbourMatrix(:,11) <= obj.MinNumPackets ); % The
                                        opposite of the trusted
43                                  obj.NeighbourMatrix = [
                                        obj.NeighbourMatrix(obj.TrustedNeighbours,:);
                                        obj.NeighbourMatrix(notTrusted,:)]; % Sort matrix
44                                  obj.numTrustedNeighbours = numel(obj.TrustedNeighbours); %
                                        Set number of trusted neighbours
45
46                                  if obj.numTrustedNeighbours == 0 % If a node do not trust
                                        anyone. Pick the best node as trusted
47                                     bestFriend = find ( (obj.NeighbourMatrix(:,11) ==
                                           max(obj.NeighbourMatrix(:,11))),1 ); % The
                                           neighbour with most received packet count is
```

```
                                picked
48                          notTrusted = obj.NeighbourMatrix(:,1) ~=
                                obj.NeighbourMatrix(bestFriend,1);
49                          obj.NeighbourMatrix = [
                                obj.NeighbourMatrix(bestFriend,:);
                                obj.NeighbourMatrix(notTrusted,:)]; % Sort matrix
50                          obj.numTrustedNeighbours = 1; % Set number of trusted
                                neighbour
51                      end
52
53                      % Adjust PPS according to the amount of nodes in the network
54                      Neighbours = obj.options.amountNodes;
55                              if Neighbours > 71 % Functio will be bigger then 10
                                    for < 71
56                                      a = 11.3505; % Found from curvefit apendix H
57                                      b = -0.0017334;
58                                      obj.PPS = a*exp(b*Neighbours);
59
60                                      if obj.PPS < 0.1
61                                              obj.PPS = 0.1;
62                                      elseif obj.PPS > 10
63                                              obj.PPS = 10;
64                                      end
65
66                              else
67                                      obj.PPS = 10;
68                              end
69                              obj.PacketCNT = 1; % Reset the packet counter
70                              obj.NeighbourMatrix(:,11) = 0; % All entries are set
                                    to zero
71
72                  case 1 % I am in the construction phase
73
74                          obj.UNC =
                                numel(find(obj.NeighbourMatrix(1:obj.numTrustedNeighbours,5)
                                == 0)); % How many undiscovered nodes can i see
75
76                          MUNC =
                                max(obj.NeighbourMatrix(1:obj.numTrustedNeighbours,2));
                                % How many nodes can my neighbour see at most
77
78                          if obj.UNC > MUNC % Can i see more undiscovered nodes
79                                  obj.relayFeature = 1; % Set me as relay
80                                  obj.Role = 1;
81                                  obj.Step = 3;
82
83                          elseif obj.UNC < MUNC % Someone else sees more undiscovered
                                neighbour nodes
84                                  obj.BeRelay = obj.NeighbourMatrix(
                                        find(obj.NeighbourMatrix(1:obj.numTrustedNeighbours,2)
                                        == MUNC ),1);
85                                  if numel( obj.BeRelay ) > 1 % More than one neighbour
                                        have a higher count
86                                      obj.BeRelay = min( obj.BeRelay); % Pick the
                                            one with lower ID
87                                  end
88
```

```
89                          elseif obj.UNC == MUNC % I see the same amout of
                                undiscovered neighbour nodes as my neighbours
90                              maybeRelay = obj.NeighbourMatrix(
                                    find(obj.NeighbourMatrix(1:obj.numTrustedNeighbours,2)
                                    == MUNC ),1); % ID of nodes with
91                              if numel( maybeRelay ) > 1 % More than one neighbour
                                    have the with the same count
92                                  maybeRelay = min( maybeRelay ); % Pick the
                                        one with lower ID
93                              end
94
95                              if maybeRelay < obj.id % He should be relays
96                                  obj.BeRelay = maybeRelay;
97
98                              elseif obj.id < maybeRelay % I should be relay
99                                  obj.relayFeature = 1;
100                                 obj.Role = 1;
101                                 obj.Step = 3;
102                             end
103                         end
104
105                 case 2 % I am in the connection phase
106                         % Is everyone done with step 1. Check for both neighbours
                                and neighbours neighbours
107                         if any(obj.NeighbourMatrix(1:obj.numTrustedNeighbours,6) <
                                2) |
                                any(obj.NeighbourMatrix(1:obj.numTrustedNeighbours,4) <
                                2)
108                             disp(['Error, node ', num2str(obj.id),' is in step 2, 
                                    but should not be']) % This is a double check
                                    that everyone is done. In case a decision
                                    conflict was not registered.
109                         end
110
111                         % Find neighbour relay CID's
112                         NR = find( obj.NeighbourMatrix(1:obj.numTrustedNeighbours,5)
                                == 1); % Entry mumber of Neighbour relays
113                         NCID = unique(obj.NeighbourMatrix(NR,3)); % NEighbour relay
                                CIDs
114
115                         % Chech for two different NCID's
116                         if numel(NCID) > 1 % I see two different CID's
117                             obj.relayFeature = 1; % I make myself relay
118                             obj.Role = 1;
119                             obj.Step = 3;
120
121                         else % I did not see two different CID. Try if i can connect
                                two different CID together with a neighbour with two
                                nodes
122                             list=
                                    obj.NeighbourMatrix(1:obj.numTrustedNeighbours,3)
                                    ~= obj.CID &
                                    obj.NeighbourMatrix(1:obj.numTrustedNeighbours,5)
                                    == 2;
123                             temp =
                                    obj.NeighbourMatrix(1:obj.numTrustedNeighbours,1);
124                             IDs = temp(list);
```

```
125
126                                 % Chech for two different NCID's
127                                 if numel(IDs) > 0 % Me and someone else sees two
                                        different CID's
128                                     obj.relayFeature = 1; % I will make myself
                                            relay
129                                     obj.Role = 1;
130                                     obj.Step = 3;
131
132                                     obj.BeRelay = IDs(1); % Set other node to
                                            relay as well
133
134                                 else % I and a neighbour can not conect any
                                        unconnected relays
135                                     obj.Step = 3; % move to next step
136                                 end
137                             end
138
139                     case 3 % I am done
140                             % Is everyone done with step 2
141                             if ~( any(obj.NeighbourMatrix(1:obj.numTrustedNeighbours,6)
                                    < 3) |
                                    any(obj.NeighbourMatrix(1:obj.numTrustedNeighbours,4) <
                                    3) )
142                                 obj.Step = 4; % All neighbours are done
143                             end
144
145                     case 4 % Setup is done
146
147             end
148         end
149
150         obj.UNC = numel(find(obj.NeighbourMatrix(1:obj.numTrustedNeighbours,5) == 0)); %
                How many undiscovered nodes can i see
151
152         obj.LNS = min( obj.NeighbourMatrix(1:obj.numTrustedNeighbours,6) ); % Update
                lowest step neighbour
153
154         % Update Role, Step %%%% This entire if statement is leftover from old verision
155         if obj.Role == 2 % Check if my dominated role is still valid
156
157                 if ~any(obj.NeighbourMatrix(1:obj.numTrustedNeighbours,5) == 1) % I do not
                        see a relay in FilteredNeighbourMatrix any more
158                     obj.Role = 0; % I am therefore not dominated any more.
159                     obj.Step = 1; % Go back and start over
160                     obj.CID = 0;
161                 end
162         end
163
164         if obj.Role == 0 % If undiscovered
165                 if any(obj.NeighbourMatrix(1:obj.numTrustedNeighbours,5) == 1) % Any
                        relays in my trusted NeighbourMatrix?
166                     obj.Role = 2; % I am dominated
167                     obj.Step = 2;
168                 end
169         end
170
```

```
171         % Check BR and see if i should be relay
172         if obj.Role == 0 || obj.Role == 2 % if i am not relays
173                 if any(obj.NeighbourMatrix(:,7) == obj.id) % The packet marked me as relay
174                         obj.relayFeature = 1; % make me a relay
175                         obj.Role = 1;
176                         obj.Step = 3;
177                 end
178         end
179
180         % Remove old BR
181         if obj.BeRelay > 0
182                 NR = find( obj.NeighbourMatrix(1:obj.numTrustedNeighbours,5) == 1);
183                 NRID = unique(obj.NeighbourMatrix(NR,1));
184                 if ismember(obj.BeRelay, NRID)
185                         obj.BeRelay = 0;
186                 end
187         end
188
189         % Update CID, only look at neighbour relays
190         if obj.Role == 1 || obj.Role == 2 % If i am dominated
191                 % Set CID to the highest CID of the neighbour relays
192                 NR = find( obj.NeighbourMatrix(1:obj.numTrustedNeighbours,5) == 1);
193                 NCID = unique(obj.NeighbourMatrix(NR,3));
194                 MaxNeighbourCID = max(NCID);
195
196                 if isempty(MaxNeighbourCID)
197                         if obj.Role == 1 % I am the first relay in my reach
198                                 obj.CID = obj.id; % If it is empty the relay takes the id of
                                        itself as its CID
199                         else % If the node is dominated it will never be empty.
200                                 disp('Dominated node have a empty MaxNeighbourCID')
201                         end
202                 elseif MaxNeighbourCID > obj.CID & obj.Role == 1
203                         obj.CID = MaxNeighbourCID; % Take the highest CID of negihbour
                                relay?
204                 elseif obj.Role == 2
205                         obj.CID = MaxNeighbourCID;
206                 end
207         end
208
209         % UNC   CID     LNS     Role    Step    BeRelay         PPS     PacketCNT
210         obj.message = [obj.UNC obj.CID obj.LNS obj.Role obj.Step obj.BeRelay obj.PPS
                obj.PacketCNT];
211
212         % Usage: createMsg(obj,ID,SRC,DST,TTL,len,message)
213         obj = createMsg(obj, obj.PacketID, obj.id, -1, 0, obj.SetupPacketLength,
                obj.message); % Generate the packet
214         obj.PacketID = obj.PacketID + 1;
215         obj.PacketCNT = obj.PacketCNT +1;
216
217         obj.idleDelay = randomtime(1, round((125000)/(obj.PPS/2))); % Decide when to send
                next packet
218 end
```